// STRYKE //

stryke — Language Reference

v0.13.0

LANGUAGE REFERENCE

every builtin · keyword · alias · extension

2026-05-10

MenkeTechnologies

Contents

Parallel Primitives 6

Shared State & Concurrency 10

Pipeline & Pipe-Forward 13

Streaming Iterators 17

List Operations 22

stryke Extensions 34

Data & Serialization 38

HTTP & Networking 47

Crypto & Encoding 52

Special Math Functions 71

Bessel / Airy / Hankel / Struve / Kelvin 73

Orthogonal Polynomials 75

Elliptic Integrals & Functions 77

Zeta / Polylog / Lerch 79

Hypergeometric 80

Modular Forms 81

Si / Ci / Ei / Li / Fresnel Integrals 82

Number Theory (Wolfram Parity) 83

Combinatorics (Wolfram Parity) 85

q-Series, Mittag-Leffler, Coulomb 86

Inverse Special Functions 87

Piecewise & Symbolic Primitives 88

Number Theory (Wolfram Parity II) 89

Combinatorial Sequences 91

Linear Algebra (Extended) 93

Polynomial Algebra 94

Distributions (Extended) 95

Mathieu / Heun 97

Wavelets 98

Graph Algorithms (Extended) 99
Number-Theoretic Sums & Constants 100
Vector Calculus 101
Optimization 102
Numerical Integration 103
Linear Algebra (Numeric) 104
Polynomial Helpers 105
Quaternions & 3D Rotations 106
Information Theory 108
Quantum Primitives 109
Statistical Mechanics 110
Optics 111
Astrodynamics 112
Time Series 113
Graph Centrality 114
Random Samplers (Extended) 115
2-D Geometry 116
Auto-Differentiation 117
Statistical Tests (R parity) 118
Distance Metrics (Distances.jl parity) 120
Multivariate / Non-Central Distributions 121
Matrix Functions 122
Adaptive ODE Solvers 123
Generalized Linear Models 124
Bootstrap & Resampling 125
Time Series Operators 126
Combinatorial Generators 127
Dynamic Programming 128
ML Metrics 129
DSP / Image Filters 130
Stochastic Processes 131
Compression / Information Complexity 132
Physics & Quantum (Extended) 133
Number Theory & Algebra (Extended) 134

CAS-Lite 135
Quadrature (Specialised) 136
Optimization (Heuristic) 137
Distributions (Tail / Mixture / Categorical) 138
Clustering & Dimensionality Reduction 140
Neural-Net Primitives 141
Time Series (Spectral) 142
Image Processing 143
Loss Functions 144
Spatial / Geographic 145
Integer Sequences 146
Graph Link & Cluster Metrics 147
3-D Geometry 148
Iterative Solvers 149
Crypto / Modular Algebra 150
Physics & Chemistry 151
MCMC & SDEs 152
Hidden Markov Models 153
Survival Analysis 154
Sequence Alignment 155
Chemistry 156
Control Theory 157
Game Theory 158
Operations Research 159
Numerical PDE 160
Bayesian Conjugate Updates 161
Quantum Computing Gates 162
Splines & Curves 163
Music & Audio 164
Astronomy & Cosmology 165
Fluid Dynamics 166
Distributions (Discrete / Circular) 167
Random Graph Generators 168
Color Science 169

Integer Sequences (Combinatorial) 170
Multiple-Testing Corrections 171
Probabilistic Distances 172
Distributions (Tail / Inflated) 173
Astrophysics & Radiation 174
Chemistry (Beer / Rate / Colligative) 175
Mixed-Strategy Game Theory 176
Computer Graphics 177
DSP (Hilbert / Cepstrum / Filter Design) 178
Image Processing (Edges / Bilateral) 179
Clustering Helpers 180
Combinatorics (Tableaux / Boustrophedon) 181
Number Theory (Tetration / Smoothness) 182
Network Coreness & Structure 183
Crypto Helpers 184
Quantum Entanglement 185
2-D Geometry (Polygons & Circles) 186
Time-Series Smoothing 187
Bioinformatics 188
Geographic / Map Projection 189
Fixed-Income Finance 190
Image Quality Metrics 191
Acoustics 192
Population Genetics 193
Epidemiology 194
Inequality Measures 195
APL / J Array Primitives 196
Plasma Physics 197
String Similarity 198
Rating Systems 199
Effect Sizes 200
Control Transient Response 201
Matrix Norms 202
Network Triad Analysis 203

Misc Inverses 204
Parallel I/O 205
File I/O 208
Strings 217
Arrays & Hashes 223
Control Flow 228
Error Handling 233
AOP / Advice 236
Declarations 238
Cluster / Distributed 241
Datetime 242
Math 246
File System 249
Process 256
Pack / Binary 260
Logging 261
Charts (SVG) 263
Audio 266
Network Utilities 267
Markup / Web Scraping 270
Date Utilities 271
Git 272
System 274
Testing 277
Test Runner (worker pool) 279
AI primitives (docs/AI_PRIMITIVES.md) 280
MCP (Model Context Protocol) 289
Web framework (stryke_web/README.md) 291
Stress / telemetry (docs/STRESS_TESTING.md) 305
PTY / expect (docs/expect-feature-idea.md) 310
Other 312

Parallel Primitives

18 topics

pmap

Parallel `map` powered by rayon's work-stealing thread pool. Every element of the input list is processed concurrently across all available CPU cores, and the output order is guaranteed to match the input order. This is the primary workhorse for CPU-bound transforms in stryke — use it whenever you have a pure function and a large list. Pass `progress => 1` to get a live progress bar on STDERR for long-running jobs.

Two equivalent surface syntaxes: • Block form — `pmap BLOCK LIST` — element bound to `_` • Bare-fn form — `pmap FUNC, LIST` — single-arg function name as first argument

```
# Block form
my @out = pmap { _ * 2 } 1:1_000_000
my @hashes = pmap sha256 @blobs, progress => 1
1:100 |> pmap { fetch("https://api.example.com/item/$_") } |> e p

# Bare-fn form (works for builtins and user-defined subs)
my @hashes = pmap sha256, @blobs, progress => 1
fn dbl = _ * 2
my @r = pmap dbl, (1:1_000_000)
```

pmaps

Streaming parallel `map` — returns a lazy iterator that processes items across all CPU cores using a persistent worker-thread pool. Unlike `pmap` which eagerly collects all results into an array, `pmaps` yields results as they complete through a bounded channel, so downstream consumers see output immediately. Output order is non-deterministic (completion order). Each worker thread reuses a single interpreter instance, making it faster than `pmap` for large inputs.

Best for: • Pipelines with `take`/`head` — avoids processing the full list • Very large inputs where holding all results in memory is impractical • Streaming pipelines where you want progressive output

```
range(0, 1e9) |> pmaps { _ * 2 } |> take 10 |> ep
t 1:1e6 pmaps { expensive(_) } ep
range(0, 1e6) |> pmaps { fetch("https://api/item/$_") } |> pgreps { _->{ok} } |> ep
```

pmap_chunked

Parallel map that groups input into contiguous batches of N elements before distributing to threads. This reduces per-item scheduling overhead when the per-element work is very cheap (e.g. a few arithmetic ops). Each thread receives a slice of N consecutive items, processes them sequentially within the batch, then returns the batch result. Use this instead of `pmap` when profiling shows rayon overhead dominates the actual computation.

```
my @out = pmap_chunked 100, { _ ** 2 } 1:1_000_000
my @parsed = pmap_chunked 50, { json_decode } @json_strings
```

pgrep

Parallel `grep` that evaluates the filter predicate concurrently across all CPU cores using rayon. The result preserves the original input order, so it is a drop-in replacement for `grep` on large lists. Best suited for predicates that do meaningful work per element — if the predicate is trivial (e.g. a single regex on short strings), sequential `grep` may be faster due to lower scheduling overhead.

Two equivalent surface syntaxes: `pgrep { BLOCK } LIST` or `pgrep FUNC, LIST`.

```
# Block form
my @matches = pgrep { /complex_pattern/ } @big_list
my @primes = pgrep { is_prime } 2:1_000_000
@files |> pgrep { -s _ > 1024 } |> e p

# Bare-fn form
fn Demo::even = _ % 2 == 0
my @e = pgrep Demo::even, 1:10  # (2,4,6,8,10)
```

pgreps

Streaming parallel `grep` — returns a lazy iterator that filters items across all CPU cores. Unlike `pgrep` which eagerly collects all matching items, `pgreps` yields matches as they are found through a bounded channel. Output order is non-deterministic (completion order). Each worker thread reuses a single interpreter instance.

```
range(0, 1e6) |> pgreps { is_prime(_) } |> take 100 |> ep
t 1:1e9 pgreps { _ % 7 == 0 } take 10 ep
```

pfor

Parallel `foreach` that executes a side-effecting block across all CPU cores with no return value. Use this when you need to perform work for each element (writing files, sending requests, updating shared state) but don't need to collect results. The block receives each element as `_`. Iteration order is non-deterministic, so the block must be safe to run concurrently.

Two equivalent surface syntaxes: `pfor { BLOCK } LIST` or `pfor FUNC, LIST`.

```
# Block form
pfor { write_report } @records
pfor { compress_file } glob("*.log")
@urls |> pfor { fetch
    p "done: $_" }

# Bare-fn form
fn task = ep "did $_"
pfor task, (1, 2, 3)
```

psort

Parallel sort that uses rayon's parallel merge-sort algorithm. Accepts an optional comparator block using `$_0`/`$_1` (or `$a`/`$b`). For large lists (10k+ elements), this significantly outperforms the sequential `sort` by splitting the array, sorting partitions in parallel, and merging. The sort is stable — equal elements retain their relative order.

```
my @sorted = psort { $_0 <=> $_1 } @big_list
my @by_name = psort { $_0->{name} cmp $_1->{name} } @records
@nums |> psort { $a <=> $b } |> e p
```

pcache

Parallel memoized map — each element is processed concurrently, but results are cached by the stringified value of `_` so duplicate inputs are computed only once. This is ideal when your input list contains many repeated values and the computation is expensive. The cache is a concurrent hash map shared across all threads, so there is no lock contention on reads after the first computation.

Two equivalent surface syntaxes: `pcache { BLOCK } LIST` or `pcache FUNC, LIST`.

```
# Block form
my @out = pcache { expensive_lookup } @list_with_dupes
my @resolved = pcache { dns_resolve } @hostnames

# Bare-fn form
my @resolved = pcache dns_resolve, @hostnames
```

preduce

Parallel tree-fold using rayon's `reduce` — splits the list into chunks, reduces each chunk independently, then merges partial results. The combining operation must be associative (e.g. `+`, `*`, `max`); non-associative ops will produce incorrect results. Much faster than sequential `reduce` on large numeric lists because the tree structure allows O(log n) merge depth across cores.

```
my $total = preduce { $_0 + $_1 } @nums
my $biggest = preduce { $_0 > $_1 ? $_0 : $_1 } @vals
my $product = preduce { $a * $b } 1:100
```

preduce_init

Parallel fold with identity value.

```
my $total = preduce_init 0, { $_0 + $_1 } @list
```

pmap_reduce

Fused parallel map + tree reduce.

```
my $sum = pmap_reduce { _*2 } { $_0 + $_1 } @list
```

pany

`pany { COND } @list` — parallel short-circuit `any`.

pfirst

`pfirst { COND } @list` — parallel first matching element.

puniq

`puniq @list` — parallel unique elements.

pflat_map

Parallel flat-map: map + flatten results. Each element produces zero or more output values via the block/function, and the outputs are concatenated in input order.

Two equivalent surface syntaxes: `pflat_map BLOCK LIST` or `pflat_map FUNC, LIST`.

```
# Block form
my @out = pflat_map expand @list

# Bare-fn form
fn expand = (_, _ * 10)
my @r = pflat_map expand, (1, 2, 3)   # (1, 10, 2, 20, 3, 30)
```

pflat_maps

Streaming parallel flat-map — like `pmaps` but flattens array results. Returns a lazy iterator.

```
range(0, 1e6) |> pflat_maps { [_, _ * 10] } |> ep
```

fan

Execute BLOCK or FUNC N times in parallel (_ = index 0..N-1). With no count, defaults to the rayon pool size (`stryke -j`).

Two equivalent surface syntaxes: • Block form — `fan N { BLOCK }` or `fan { BLOCK }` • Bare-fn form — `fan N, FUNC` or `fan FUNC`

```
# Block form
fan 8 { work(_) }
fan { work(_) } progress => 1

# Bare-fn form
fn tick = ep "tick $_"
fan 8, tick
fan tick, progress => 1   # uses pool size
```

fan_cap

Like `fan` but captures return values in index order. Two surface syntaxes: `fan_cap N { BLOCK }` or `fan_cap N, FUNC`.

```
# Block form
my @results = fan_cap 8 { compute(_) }

# Bare-fn form
fn compute = _ * _
my @squares = fan_cap 8, compute
```

Shared State & Concurrency

11 topics

mysync

Declare shared variables for parallel blocks (`Arc<Mutex>`).

```
mysync $counter = 0
fan 10000 { $counter++ }   # always exactly 10000
mysync @results
mysync %histogram$1

Compound ops (`++`, `+=`, `.=`, `|=`, `&=`) are fully atomic.
```

async

`async { BLOCK }` — schedule a block for execution on a background worker thread and return a task handle immediately.

The block begins executing as soon as a thread is available in stryke's global rayon thread pool, while the calling code continues without blocking. To retrieve the result, pass the task handle to `await`, which blocks until the task completes and returns its value. If the block panics, the panic is captured and re-raised at the `await` call site. Use `async` for fire-and-forget background work, overlapping I/O with computation, or launching multiple independent tasks that you later join. For structured fan-out with index-based parallelism, prefer `fan` or `fan_cap` instead.

```
my $task = async { long_compute() }
do_other_work()
my $val = await $task

# Overlapping multiple fetches:
my @tasks = map { async { fetch("https://api.example.com/$_") } } 1:10;
my @results = map { await _ } @tasks
```

spawn

`spawn { BLOCK }` — Rust-style alias for `async`; schedules a block on a background thread and returns a joinable task handle.

This is identical to `async` in every respect — same thread pool, same semantics, same `await` for joining. The name exists for developers coming from Rust's `tokio::spawn` or `std::thread::spawn` who find `spawn` more natural. Use whichever reads better in your code; mixing `async` and `spawn` in the same program is perfectly fine since they share the same underlying pool.

```
my $task = spawn { expensive_io() }
my $val = await $task

my @handles = map { spawn { process } } @items;
my @out = map { await _ } @handles
```

await

`await $task` — block the current thread until an async/spawn task completes and return its result value.

If the background task has already finished by the time `await` is called, the result is returned immediately with no scheduling overhead. If the task panicked, `await` re-raises the panic as a die in the calling thread, preserving the original error message and backtrace. You can `await` a task exactly once; calling `await` on an already-joined handle is a fatal error. For waiting on multiple tasks, simply map over the handles — stryke does not yet provide a `join_all` primitive, but `map await @tasks` achieves the same effect.

```
my $task = async { 42 }
my $result = await $task              # 42

# Await with error handling:
my $t = spawn { die "oops" }
eval { await $t }                     # $@ eq "oops"
```

pchannel

`pchannel(N)` — create a bounded multi-producer multi-consumer (MPMC) channel with capacity N, returning a `($tx, $rx)` pair.

The transmitter `$tx` supports `->send($val)` which blocks if the channel is full (backpressure). The receiver `$rx` supports `->recv` which blocks until a value is available, and `->try_recv` which returns `undef` immediately if the channel is empty. Both ends can be cloned and shared across threads — clone `$tx` with `$tx->clone` to create additional producers, or clone `$rx` for additional consumers. When all transmitters are dropped, `->recv` on the receiver returns `undef` to signal completion. Use `pchannel` to build producer-consumer pipelines, rate-limited work queues, or to communicate between `async`/`spawn` tasks.

```
my ($tx, $rx) = pchannel(100)
async { $tx->send(_) for 1:1000
    undef $tx }
while (defined(my $val = $rx->recv)) {
    p $val
}

# Multiple producers:
my ($tx, $rx) = pchannel(50)
for my $i (1:4) {
    my $t = $tx->clone
    spawn { $t->send("from worker $i: _") for 1:100 }
}
undef $tx  # drop original so channel closes when workers finish
```

pselect

`pselect(@channels)` — wait on multiple `pchannel` receivers.

barrier

`barrier(N)` — create a synchronization barrier that blocks until exactly N threads have arrived at the wait point.

Each thread calls `$b->wait` and is suspended until all N participants have reached the barrier, at which point all are released simultaneously. This is useful for coordinating phased parallel algorithms where all workers must complete step K before any worker begins step K+1. The barrier is reusable — after all threads are released, it resets and can be waited on again. Internally backed by a Rust `std::sync::Barrier` for zero-overhead synchronization.

```
my $b = barrier(4)
for my $i (0:3) {
    spawn {
        setup_phase($i)
        $b->wait                      # all 4 threads sync here
        compute_phase($i)
    }
}
```

ppool

`ppool(N, fn { ... })` — create a persistent thread pool of N worker threads, each running the provided subroutine in a loop.

The pool is typically paired with a `pchannel`: workers pull items from the channel's receiver, process them, and optionally send results to an output channel. Unlike `pmap` which is a one-shot parallel transform, `ppool` keeps threads alive for the lifetime of the pool, making it ideal for long-running server-style workloads, background drain loops, or scenarios where thread startup cost would dominate short-lived `pmap` calls. Workers exit when their input channel is closed (all transmitters dropped). The pool object supports `->join` to block until all workers have finished.

```
my ($tx, $rx) = pchannel(100)
my $pool = ppool 4, fn {
    while (defined(my $job = $rx->recv)) {
        process($job)
    }
}
$tx->send(_) for @work_items
undef $tx                           # signal completion
$pool->join                         # wait for drain
```

deque

`deque LIST` — create a double-ended queue initialized with the given elements.

A deque supports efficient O(1) insertion and removal at both ends via `->push_front($val)`, `->push_back($val)`, `->pop_front`, and `->pop_back`. It also supports `->len` and iteration. Internally backed by a Rust `VecDeque`, it is ideal for sliding window algorithms, BFS traversals, or any scenario where you need fast access to both ends of a sequence.

```
my $dq = deque(1, 2, 3)
$dq->push_front(0)
$dq->push_back(4)
p $dq->pop_front                    # 0
```

heap

`heap LIST` — create a min-heap (priority queue) from the given elements.

Elements are heapified on construction so that `->pop` always returns the smallest element in O(log n) time. `->push($val)` inserts a new element, also in O(log n). The heap supports `->peek` to inspect the minimum without removing it, and `->len` for the current size. Internally backed by a Rust `BinaryHeap` (inverted for min-heap semantics), it is the go-to structure for top-K queries, Dijkstra, and merge-K-sorted-lists problems.

```
my $h = heap(5, 3, 8, 1)
p $h->pop                           # 1 (smallest first)
p $h->peek                          # 3 (next smallest)
$h->push(0)
```

set

`set LIST` — create a set (unique unordered collection) from the given elements.

Duplicate values are collapsed on construction so the set contains each value exactly once. The set object provides `->contains($val)` for O(1) membership testing, plus `->union($s)`, `->intersection($s)`, `->difference($s)`, and `->len` methods. Internally backed by a Rust `HashSet` for performance. Use `to_set` to convert an existing iterator into a set.

```
my $s = set(1, 2, 3, 2, 1)
p $s->contains(2)                   # 1
p $s->len                           # 3
my $both = $s->union(set(3, 4, 5))
```

Pipeline & Pipe-Forward

12 topics

`|>`

Pipe-forward operator — threads LHS as first argument of RHS call.

```
"hello" |> uc |> rev |> p              # OLLEH
1:10 |> grep _ > 5 |> map _ * 2 |> e p
$url |> fetch_json |> json_jq '.name' |> p
"hello world" |> s/world/perl/ |> p     # hello perl
```

Zero runtime cost (parse-time desugaring). Binds looser than `||`, tighter than `?:`.

`thread`

Thread-first macro — chain stages where the threaded value is injected as the first argument (like Clojure's `->`).

```
~> @data grep { _ > 5 } map { _ * 2 } sort { $_0 <=> $_1 } |> join "," |> p
~> " hello " tm uc rv lc ufc sc cc kc tj p  # short aliases
fn div = _0 / _1
~> 10 div(2) p                              # div(10, 2) = 5
~> 10 div(_, 2) p                           # explicit _ position: div(10, 2) = 5
```

Spellings: `~>`, `thread`, `t`. Stages: bare function (`uc`, `tm`, …), function with block (`map { … }`, `grep { … }`), `name(args)` call where `_` is the threaded-value placeholder, or `>{}` anonymous block. `|>` terminates the macro.

`->>`

Thread-last macro — chain stages where the threaded value is injected as the last argument (like Clojure's `->>`).

```
fn div = _0 / _1
~>> 10 div(2) p                             # div(2, 10) = 0.2
->> 10 div(_, 2) p                          # explicit _ overrides: div(10, 2) = 5

# Thread-last is useful for list-consuming functions:
fn first_n { @_[0..$_[-1]-1] }              # first_n(list, n)
~>> (1,2,3,4,5) first_n(3) p                # first_n(1,2,3,4,5, 3) = (1,2,3)
```

Spellings: `~>>`, `->>`. The explicit _ placeholder always takes precedence.

`~s>`

Per-item streaming thread macro — each stage runs as an independent worker thread connected to the next via a bounded channel, so items flow through the pipeline concurrently without materializing intermediate arrays.

```
~s> [1, 2, 3, 4, 5] map { _ * 10 } map { _ + 1 }
~s> @lines map { json_decode } grep { _->{ok} } map { _->{val} }
~s> gen { yield_lines "huge.log" } map { parse } grep { /ERROR/ }
```

Use `~s>` when stages have differing latencies (I/O + CPU mix) so faster workers can pull ahead. Output order is non-deterministic. The macro's final value is the count of items the last stage emitted. `~s>>` is the thread-last variant.

See `~p>` for chunk-parallel (vs per-item) execution and `par_pipeline` for the verbose builder form.

~p>

Chunk-parallel thread macro — splits the input into chunks, runs the entire pipeline on each chunk in parallel via rayon, and auto-merges chunk results. Equivalent to `par_reduce { stage1 |> stage2 |> ... } SRC`.

```
~p> c("**.rs") letters freq                          # histogram across all .rs
~p> @big map { _ * 2 } sum                           # parallel map+sum
~p> $huge letters freq ||> values |> sum              # boundary: terse
~p> $huge letters freq |then| values |> sum           # boundary: english
```

The auto-merger picks a strategy by inspecting the first chunk's result type: hash-of-numbers merges key-wise with `+` (canonical histogram), numeric scalars sum, arrays concat, strings concat. Use `||>` or `|then|` to switch back to sequential `~>` mode for stages that operate on the merged whole. `|>` also terminates a `~p>` macro. `~p>>` is the thread-last variant.

par

Generic parallel-chunk thread-stage — only valid inside `~>`/`~>>`. Splits the threaded value into chunks, runs `BLOCK` on each chunk in parallel, and flattens chunk outputs into a single list.

```
my @a = ~> "Hello, World 123" par { letters }    # H e l l o W o r l d
~> @huge par { map { _ ** 2 } |> sum } |> sum p   # parallel sum-of-squares
```

Each chunk gets a fresh sub-VM with the chunk bound to `_` and `$_[0]`. List-returning ops keep their values; the runtime auto-flattens. Use `par_reduce` when you need a final merge step rather than flat concat. `par { … }` is *only* a thread-stage; calling it as a top-level expression is an undefined-sub error.

par_reduce

Chunk-extract-merge thread-stage. The single-block form runs `extract` per chunk and auto-merges results by type (hash<num>⍰key-wise add, number⍰sum, array⍰concat, string⍰concat). The two-block form binds `$a`/`$b` in the explicit reducer.

```
~> "abc def abc" par_reduce { letters |> freq }      # canonical histogram merge
~> @nums par_reduce { sum }                          # parallel sum
~> "abcdefgh" par_reduce { length(_) } { $a + $b }   # explicit pairwise reducer
```

Below the chunk threshold the runtime falls back to single-chunk eval; the result must still match the bare extract block's output. Multi-chunk paths require the merger (auto or explicit) to be associative and commutative for correct results.

||>

Boundary marker that terminates the chunk-parallel section of a `~p>` macro and switches the merged result into a sequential `~>` continuation.

```
~p> "abc def abc" letters freq ||> values |> sum     # 9
~p> "abc def abc" letters freq |then| values |> sum  # 9 — readable form
~p> @data map { _ * 2 } sum ||> sqrt p               # parallel sum, sequential sqrt
```

Both spellings are byte-equivalent; `||>` is terse, `|then|` reads in English. Use either when you need a post-merge stage that operates on the whole reduced value rather than per-chunk. `|>` also terminates a `~p>` macro.

pipeline

`pipeline(@list)` — wrap a list (or iterator) in a lazy pipeline object supporting chained `->map`, `->filter`, `->take`, `->skip`, `->flat_map`, `->tap`, and other transforms that execute zero work until a terminal method is called.

No intermediate lists are allocated between stages — each element flows through the full chain one at a time, making pipelines memory-efficient even on very large or infinite inputs. Terminal methods include `->collect` (materialize to list), `->reduce { ... }` (fold), `->for_each { ... }` (side-effect iteration), and `->count`. Pipelines compose naturally with stryke's `|>` operator: you can feed `pipeline(...)` output into further `|>` stages or vice versa. Use `pipeline` when you want explicit method-chaining style rather than the flat `|>` pipe syntax — both compile to the same lazy evaluation engine.

```
my @out = pipeline(@data)
  ->filter { _ > 0 }
  ->map { _ * 2 }
  ->take(10)
  ->collect

pipeline(1:1_000_000)
  ->filter { _ % 3 == 0 }
  ->map { _ ** 2 }
  ->take(5)
  ->for_each { p _ }                # 9 36 81 144 225

my $sum = pipeline(@scores)
  ->filter { _ >= 60 }
  ->reduce { $_0 + $_1 }
```

par_pipeline

`par_pipeline(source => \@data, stages => [...], workers => N)` — build a multi-stage parallel pipeline where each stage's map/filter block runs concurrently across N worker threads.

Unlike `pmap` which parallelizes a single transform, `par_pipeline` lets you define a sequence of named stages — each with its own block — that execute in parallel while preserving input order in the final output. Internally, stryke partitions the source into chunks, distributes them across workers, and pipelines the stages so that stage 2 can begin processing a chunk as soon as stage 1 finishes it, overlapping computation across stages. This is ideal for multi-step ETL workloads where each step is CPU-bound and the data volume is large. The `workers` parameter defaults to the number of logical CPUs.

```
my @results = par_pipeline(
    source  => \@raw_records,
    stages  => [
        { name => "parse",     map => fn { json_decode } },
        { name => "transform", map => fn { enrich } },
        { name => "validate",  filter => fn { _->{valid} } },
    ],
    workers => 8,
)
```

par_pipeline_stream

`par_pipeline_stream(source => ..., stages => [...], workers => N)` — streaming variant of `par_pipeline` that connects stages via bounded `pchannel` queues instead of materializing intermediate arrays.

Each stage runs as an independent pool of workers, pulling from an input channel and pushing to an output channel. This gives true pipelined parallelism: stage 1 workers produce items while stage 2 workers consume them concurrently, bounded by channel capacity to prevent memory blowup. The streaming design makes this suitable for infinite or very large data sources (file streams, network feeds) where materializing the full dataset between stages is impractical. Results are emitted in arrival order by default; pass `ordered => 1` to reorder them to match input order at the cost of buffering.

```
par_pipeline_stream(
    source  => fn { while (my $line = <STDIN>) { yield $line } },
    stages  => [
        { name => "parse", map => fn { json_decode } },
        { name => "score", map => fn { compute_score } },
    ],
    workers => 4,
    on_item => fn { p _ },           # process results as they arrive
)
```

collect

`collect ITERATOR` — materialize a lazy iterator or pipeline into a concrete list.

Pipeline stages in stryke are lazy: chaining `|> map {...} |> grep {...}` builds up a deferred computation without consuming any elements. Calling `collect` forces evaluation and returns all results as a regular Perl list. This is the standard way to terminate a lazy chain when you need the full result in an array. Without `collect`, the iterator is consumed element-by-element by `e`, `take`, or other streaming sinks.

```
my @out = range(1,5) |> map { _ * 2 } |> collect
gen { yield _ for 1:3 } |> collect |> e p
my @data = stdin |> grep /INFO/ |> collect
```

Streaming Iterators

24 topics

maps

`maps { BLOCK } LIST` — the lazy, streaming counterpart of `map`. Instead of materializing the entire output list, `maps` returns a pull iterator that evaluates the block on demand as downstream consumers request values. This makes it ideal for `|>` pipeline chains, especially when combined with `take`, `greps`, or `collect`. Use `maps` over `map` when working with large ranges, infinite sequences, or when you want to short-circuit processing early with `take`. Memory usage is constant regardless of input size.

```
1:10 |> maps { _ * 3 } |> take 4 |> e p
# 3 6 9 12
1:1_000_000 |> maps { _ ** 2 } |> greps { _ > 100 } |> take 3 |> e p
```

greps

`greps { BLOCK } LIST` — the lazy, streaming counterpart of `grep`. Returns a pull iterator that only evaluates the predicate as elements are requested downstream. This is the preferred filtering function in `|>` pipelines because it avoids materializing intermediate lists. Combine with `take` to short-circuit early, or with `maps` and `collect` for full lazy pipelines. The block receives `_` just like `grep`.

```
1:100 |> greps { _ % 7 == 0 } |> take 3 |> e p
# 7 14 21
@lines |> greps { /ERROR/ } |> maps { tm } |> e p
```

filter

`filter` (alias `fi`) `{ BLOCK } LIST` — stryke-native lazy filter that returns a pull iterator, functionally identical to `greps` but named for familiarity with Rust/Ruby/JS conventions. Use `filter` or `greps` interchangeably in `|>` chains; both are streaming and both set `_` in the block. The result must be consumed with `collect`, `e p`, `foreach`, or another terminal. Prefer `filter` when writing stryke-idiomatic code; prefer `grep`/`greps` when porting from Perl.

```
my @big = 1:1000 |> filter { _ > 990 } |> collect
@big |> e p   # 991 992 993 994 995 996 997 998 999 1000
1:50 |> fi { _ % 2 } |> take 5 |> e p   # 1 3 5 7 9
```

tap

`tap { BLOCK } LIST` — execute a side-effecting block for each element, then pass the element through unchanged.

The return value of the block is ignored; the original element is always forwarded to the next pipeline stage. This makes `tap` ideal for injecting logging, debugging, or metrics collection into the middle of a pipeline without altering the data flow. It is fully streaming and preserves element order.

```
1:5 |> tap { log_debug "saw: $_" } |> map { _ * 2 } |> e p
@files |> tap { p "processing: $_" } |> map slurp |> e p
```

peek

`peek ITERATOR` — inspect the next element of an iterator without consuming it.

The peeked value is buffered internally so that the next call to `->next` or pipeline pull returns the same element. This is useful for lookahead parsing, conditional branching on the next value, or implementing `take_while`-style logic manually. Calling `peek`

multiple times without advancing the iterator returns the same value each time. Works with any stryke iterator including `gen`, `range`, `stdin`, and pipeline results.

```
my $g = gen { yield _ for 1:5 }
p peek $g                          # 1 (not consumed)
p $g->next                         # 1
p peek $g                          # 2
```

tee

`tee FILE, ITERATOR` — write each element to a file as a side effect while passing it through the pipeline.

Every element that flows through `tee` is appended as a line to the specified file, and the element itself continues downstream unchanged. The file is opened once on first element and closed when the iterator is exhausted. This is the stryke equivalent of the Unix `tee` command, useful for auditing or logging intermediate pipeline results to disk.

```
1:10 |> tee "/tmp/log.txt" |> map { _ * 2 } |> e p
stdin |> tee "/tmp/raw.log" |> grep /ERROR/ |> e p
```

take

`take N, LIST` / `head N, LIST` / `hd N, LIST` — return at most the first N elements.

In streaming mode, `take` pulls exactly N elements from the upstream iterator and then stops, so it short-circuits infinite or very large sources efficiently. This makes it safe to write `range(1, 1_000_000) |> take 5` without allocating a million-element list. If the source has fewer than N elements, all of them are returned. The `head` and `hd` aliases mirror Unix `head` semantics.

```
1:100 |> take 5 |> e p              # 1 2 3 4 5
my @top = hd 3, @sorted
stdin |> hd 10 |> e p               # first 10 lines
```

tail

`tail N, @list` — returns the last N elements of the list. If N exceeds the list length, the entire list is returned. The alias `tl` is also available. This is the complement of `head`/`take`. In stryke, it works both as a function call and in `|>` pipelines.

```
my @t = tail 2, 1:5
@t |> e p   # 4 5
1:100 |> tl 3 |> e p  # 98 99 100
```

drop

`drop N, LIST` / `skip N, LIST` / `drp N, LIST` — skip the first N elements and return the rest.

This operation is fully streaming: when used in a pipeline, the first N elements are consumed and discarded without buffering, and all subsequent elements flow through to the next stage. If the list contains fewer than N elements, the result is empty. The `skip` and `drp` aliases exist for readability in different contexts — all three compile to the same internal op.

```
1:10 |> drop 3 |> e p               # 4 5 6 7 8 9 10
my @rest = drp 2, @data
stdin |> skip 1 |> e p               # skip header line
```

take_while

`take_while { COND } LIST` — emit leading elements while the predicate returns true, then stop.

The block receives each element as `_`. Elements are emitted as long as the predicate holds; the moment it returns false, the pipeline terminates immediately without consuming further input. This short-circuit behavior makes `take_while` efficient on infinite iterators and large streams. It is the complement of `drop_while`.

```
1:10 |> take_while { _ < 5 } |> e p # 1 2 3 4
stdin |> take_while { !/^END/ } |> e p
```

drop_while

`drop_while { COND } LIST` — skip leading elements while the predicate returns true, then emit everything after.

The block receives each element as `_`. Once the predicate returns false for the first time, that element and all remaining elements pass through unconditionally — the predicate is never consulted again. This is streaming: elements are tested one at a time without buffering. Useful for skipping headers, preamble, or sorted prefixes in data streams.

```
1:10 |> drop_while { _ < 5 } |> e p # 5 6 7 8 9 10
@log |> drop_while { /^ #/ } |> e p    # skip comment header
```

reject

`reject { BLOCK } LIST` — stryke-native streaming inverse of `filter`/`greps`. It keeps only the elements for which the block returns false. This reads more naturally than `filter { !(...) }` when the condition describes what you want to exclude rather than what you want to keep. Like `filter` and `greps`, it returns a lazy iterator suitable for `|>` chains. The block receives `_` as the current element.

```
1:10 |> reject { _ % 3 == 0 } |> e p
# 1 2 4 5 7 8 10
@files |> reject { /\.bak$/ } |> e p   # skip backups
```

compact

`compact` (alias `cpt`) — stryke-native streaming operator that removes `undef` and empty-string values from a list or iterator. This is a common data-cleaning step when dealing with parsed input, optional fields, or split results that produce empty segments. It is equivalent to `greps { defined(_) && _ ne "" }` but more concise and faster because the check is inlined in Rust. Numeric zero and the string `"0"` are preserved since they are defined and non-empty.

```
my @raw = (1, undef, "", 2, undef, 3)
@raw |> compact |> e p   # 1 2 3
split(/,/, "a,,b,,,c") |> cpt |> e p   # a b c
```

concat

`concat` (alias `chain`) — stryke-native streaming operator that concatenates multiple lists or iterators into a single sequential iterator. Pass array references and they will be yielded in order without copying. This is useful for merging data from multiple sources into a unified pipeline. The operation is lazy: each source is drained in turn, so memory usage stays proportional to the largest single element, not the total.

```
my @a = 1:3
my @b = 7:9
concat(\@a, \@b) |> e p   # 1 2 3 7 8 9
my @c = ("x")
concat(\@a, \@b, \@c) |> maps { uc } |> e p
```

enumerate

`enumerate ITERATOR` / `en ITERATOR` — yield `[$index, $item]` pairs from a streaming iterator.

Each element is wrapped as `[$index, $item]` where the index starts at 0 and increments for each element. Unlike `with_index` which returns `[item, index]`, `enumerate` uses the Rust/Python convention of `[index, item]`. This is a streaming operation: the index counter is maintained lazily as elements flow through the pipeline.

```
stdin |> en |> e { p "_->[0]: _->[1]" }
1:5 |> en |> e { p "_->[0]: _->[1]" }
@lines |> en |> grep { _->[0] < 10 } |> e { p _->[1] }
```

chunk

`chunk N, ITERATOR` / `chk N, ITERATOR` — group elements into N-sized arrayrefs as they stream through.

Elements are buffered until N are collected, then the group is emitted as a single arrayref. The final chunk may contain fewer than N elements if the source is not evenly divisible. This is fully streaming: only one chunk is held in memory at a time, making it safe for large or infinite iterators. Use `chunk` for batching work (e.g., bulk database inserts) or formatting output into rows.

```
1:9 |> chk 3 |> e { p join ",", @_ }
# 1,2,3  4,5,6  7,8,9
stdin |> chk 100 |> e { bulk_insert(@_) }
```

dedup

`dedup ITERATOR` / `dup ITERATOR` — drop consecutive duplicate elements from a stream.

Only adjacent duplicates are removed: if the same value appears later after an intervening different value, it is emitted again. Comparison is string-based by default. This is a streaming operation that holds only the previous element in memory, so it works on infinite iterators. For global deduplication across the entire stream, use `distinct` instead.

```
1,1,2,2,3,1,1 |> dedup |> e p        # 1 2 3 1
@sorted |> dedup |> e p              # like uniq(1)
stdin |> dedup |> e p                 # collapse repeated lines
```

distinct

`distinct LIST` — remove duplicate elements, preserving first-occurrence order (alias for `uniq`).

Each element is compared as a string; the first time a value is seen it is emitted, and all subsequent occurrences are silently dropped. When used in a pipeline the deduplication state is maintained across streamed chunks, so `distinct` works correctly on lazy iterators and generators. This is a stryke built-in backed by a hash set internally, so it runs in O(n) time regardless of list size.

```
my @u = distinct(3,1,2,1,3)           # (3,1,2)
1,2,2,3,3,3 |> distinct |> e p        # 1 2 3
stdin |> distinct |> e p              # unique lines
```

flatten

`flatten LIST` / `fl LIST` — recursively flatten nested arrayrefs into a single flat list.

Flatten walks every element: scalars pass through unchanged, arrayrefs are opened and their contents are recursively flattened, so arbitrarily deep nesting is handled in one call. In pipeline mode (`|> fl`) it streams element-by-element, so you can chain it with `map`, `grep`, or `take` without materializing the entire intermediate list. The `fl` alias keeps pipeline chains concise.

```
my @flat = flatten([1,[2,3]],[4])     # (1,2,3,4)
[1,[2,[3,4]]] |> fl |> e p            # 1 2 3 4
@nested |> fl |> grep { _ > 0 } |> e p
```

with_index

`with_index LIST` / `wi LIST` — pair each element with its 0-based index as `[$item, $index]`.

Each element is wrapped in a two-element arrayref where `_->[0]` is the original value and `_->[1]` is its position. This is useful when you need positional information during a map or grep without maintaining a manual counter. Note the order is `[item, index]`, which differs from `enumerate` which yields `[index, item]`.

```
qw(a b c) |> wi |> e { p "_->[1]: _->[0]" }
# 0: a  1: b  2: c
@data |> wi |> grep { _->[1] % 2 == 0 } |> e { p _->[0] }
```

first_or

`first_or DEFAULT, LIST` — return the first element of the list, or DEFAULT if the list is empty.

This is a streaming terminal: it pulls exactly one element from the upstream iterator and returns it, or returns the default value if the iterator is exhausted. It never buffers the entire list. This is especially useful at the end of a `grep` or `map` pipeline where you need a safe fallback instead of `undef` when no match is found.

```
my $v = first_or 0, @maybe_empty
my $x = grep { _ > 99 } @nums |> first_or -1
stdin |> grep /^ERROR/ |> first_or "(none)" |> p
```

range

`range(START, END [, STEP])` — create a lazy integer iterator from START to END with an optional step.

The range is inclusive on both ends. When START > END and no step is given, stryke automatically counts downward. An explicit STEP controls the increment and must be negative when counting down, or the range will be empty. The iterator is fully lazy: no list is allocated, and elements are generated on demand, making `range(1, 1_000_000)` as cheap to create as `range(1, 5)`. Combine with `|>` to feed into streaming pipelines.

```
range(1, 5) |> e p                  # 1 2 3 4 5
range(5, 1) |> e p                  # 5 4 3 2 1
range(0, 10, 2) |> e p              # 0 2 4 6 8 10
range(10, 0, -2) |> e p             # 10 8 6 4 2 0
```

stdin

`stdin` — return a streaming iterator over lines read from standard input.

Each call to the iterator reads one line from STDIN, strips the trailing newline, and yields it. The iterator terminates at EOF. Because it is lazy, combining `stdin` with `take`, `grep`, or `first_or` processes only as many lines as needed — the rest of STDIN is left unconsumed. This is the idiomatic stryke way to build Unix-style filters.

```
stdin |> grep /error/i |> e p
stdin |> take 5 |> e p
stdin |> en |> e { p "_->[0]: _->[1]" }
```

nth

`nth N, LIST` — return the Nth element using 0-based indexing.

When used in a pipeline, `nth` consumes and discards the first N elements, returns the next one, and stops — so it short-circuits on infinite iterators. On a plain list, it is equivalent to `$list[N]` but works as a function call for pipeline composition. Returns `undef` if the list has fewer than N+1 elements.

```
my $third = nth 2, @data
1:10 |> nth 4 |> p                  # 5
stdin |> nth 0 |> p                  # first line
```

List Operations

62 topics

map

`map BLOCK LIST` — evaluates the block for each element of the list, setting `_` to the current element, and returns a new list of all the block's return values. This is the eager version: it consumes the entire input list and produces the entire output list before returning. Use `map` when you need the full result array or when the input is small. For large or infinite sequences, prefer `maps` (the streaming variant). The block can return zero, one, or multiple values per element, making `map` useful for both transformation and flattening. Use `{ |$var| body }` to name the block parameter instead of `_`.

```
my @sq = map { _ ** 2 } 1:5
@sq |> e p   # 1 4 9 16 25
map { |$n| $n * $n }, 1:5   # named param
my @pairs = map { (_, _ * 10) } 1:3
@pairs |> e p   # 1 10 2 20 3 30
```

grep

`grep { BLOCK } LIST` — filters the list, returning only elements for which the block evaluates to true. The current element is available as `_`. This is the eager version: it processes the entire list and returns a new list. It is Perl-compatible and works exactly like Perl's builtin `grep`. For streaming/lazy filtering in `|>` pipelines, use `greps` or `filter` instead. Use `{ |$var| body }` to name the block parameter instead of `_`.

```
my @evens = grep { _ % 2 == 0 } 1:10
@evens |> e p   # 2 4 6 8 10
grep { |$x| $x > 3 }, 1:6   # named param
my @long = grep { length(_) > 3 } @words
```

sort

`sort [BLOCK] LIST` — returns a new list sorted in ascending order. Without a block, `sort` compares elements as strings (lexicographic). Pass a comparator block using `$_0` and `$_1` (stryke style) or `$a` and `$b` (classic Perl) to control ordering: use `<=>` for numeric and `cmp` for string comparison. The sort is stable in stryke, meaning equal elements preserve their original relative order. For descending order, reverse the operands in the comparator. Use `{ |$x, $y| body }` to name the two comparator params. Works naturally in `|>` pipelines.

```
my @nums = (3, 1, 4, 1, 5)
my @asc = sort { $_0 <=> $_1 } @nums
@asc |> e p   # 1 1 3 4 5
sort { |$x, $y| $y <=> $x }, @nums   # named params
my @desc = sort { $_1 <=> $_0 } @nums
@desc |> e p  # 5 4 3 1 1
@nums |> sort |> e p   # string sort in pipeline
```

reverse

`reverse LIST` — in list context, return the elements in reverse order; in scalar context, reverse the characters of a string.

The dual nature of `reverse` is context-dependent: `reverse @array` flips element order, while `scalar reverse $string` (or just `reverse $string` in scalar context) reverses character order. In stryke, the string form is Unicode-aware, correctly reversing multi-byte characters rather than individual bytes. In `|>` pipelines, use the `t rev` shorthand for a concise streaming string reverse. `reverse` does not modify the original — it always returns a new list or string.

```
p reverse "hello"       # olleh
my @r = reverse 1:5   # (5,4,3,2,1)
@r |> e p               # 5 4 3 2 1
"abc" |> t rev |> p     # cba
```

reduce

`reduce { $_0 OP $_1 } @list` — performs a sequential left fold over the list. The first two elements are passed as `$_0` and `$_1` to the block; the result becomes `$_0` for the next iteration. The traditional `$a`/`$b` names are also supported for Perl compatibility. Use `{ |$acc, $val| body }` to name the two params. Returns `undef` for an empty list and the single element for a one-element list. For a fold with an explicit initial value, see `fold`.

```
my $fac = reduce { $_0 * $_1 } 1:6
p $fac   # 720
reduce { |$acc, $val| $acc + $val }, 1:10   # 55 (named params)
my $longest = reduce { length($_0) > length($_1) ? $_0 : $_1 } @words
```

fold

`fold { $_0 OP $_1 } INIT, @list` — left fold with an explicit initial accumulator value. The initial value is passed as the first `$_0`, and each list element arrives as `$_1`. Unlike `reduce`, `fold` never returns `undef` for an empty list — it returns the initial value instead. Both `$a`/`$b` and `$_0`/`$_1` are supported in the block. Use `fold` when you need a guaranteed starting point for the accumulation.

```
my $total = fold { $_0 + $_1 } 100, 1:5
p $total   # 115
my $csv = fold { "$_0,$_1" } "", @fields
```

reductions

`reductions { $_0 OP $_1 } @list` — returns the running (cumulative) results of a left fold, also known as a scan or prefix-sum. Each element of the output is the accumulator state after processing the corresponding input element. The output list has the same length as the input. Both `$_0`/`$_1` and `$a`/`$b` naming conventions are supported.

```
my @pfx = reductions { $_0 + $_1 } 1:4
@pfx |> e p   # 1 3 6 10
1:5 |> reductions { $_0 * $_1 } |> e p  # 1 2 6 24 120
```

all

`all { COND } @list` — returns true (1) if every element in the list satisfies the predicate, false ("") otherwise. The block receives each element as `_` and should return a boolean. Short-circuits on the first failing element, so it never evaluates more than necessary. Works with `|>` pipelines and accepts bare lists or array variables.

```
my @nums = 2, 4, 6, 8
p all { _ % 2 == 0 } @nums   # 1
1:100 |> all { _ > 0 } |> p  # 1
```

any

`any { COND } @list` — returns true (1) if at least one element satisfies the predicate, false ("") if none do. The block receives each element as `_`. Short-circuits on the first match, making it efficient even on large lists. `any` is a stryke global builtin and can be used in `|>` pipelines.

```
my @vals = 1, 3, 5, 8
p any { _ > 7 } @vals   # 1
1:1000 |> any { _ == 42 } |> p  # 1
```

none

`none { COND } @list` — returns true (1) if no element satisfies the predicate, false ("") if any element matches. Logically equivalent to `!any { COND } @list` but reads more naturally in guard clauses. Short-circuits on the first match. Useful for validation checks where you want to assert the absence of a condition.

```
my @words = ("cat", "dog", "bird")
p none { /z/ } @words   # 1
p "all non-negative" if none { _ < 0 } @vals
```

first

`first { COND } @list` — returns the first element for which the block returns true, or `undef` if no element matches. The alias `fst` is also available. Short-circuits immediately upon finding a match, so only the minimum number of elements are tested. Ideal for searching sorted or unsorted lists when you need a single result.

```
my $f = first { _ > 10 } 3, 7, 12, 20
p $f   # 12
1:1_000_000 |> first { _ % 9999 == 0 } |> p  # 9999
```

min

`min @list` — returns the numerically smallest value from a list. Compares all elements using numeric (`<=>`) comparison, so stringy values are coerced to numbers. Returns `undef` for an empty list. In stryke, `min` is a global built-in (no import needed) and works directly in `|>` pipelines.

```
p min(5, 3, 9, 1)   # 1
my @temps = (72.1, 68.5, 74.3)
@temps |> min |> p  # 68.5
```

max

`max @list` — returns the numerically largest value from a list. Compares all elements using numeric (`<=>`) comparison. Returns `undef` for an empty list. Like `min`, this is a stryke built-in available without imports and works in `|>` pipelines. Combine with `map` to extract max values from complex structures.

```
p max(5, 3, 9, 1)   # 9
1:100 |> map { _ ** 2 } |> max |> p  # 10000
```

sum

`sum @list` returns the numeric sum of all elements. Returns `undef` for an empty list. `sum0` is identical except it returns `0` for an empty list, which avoids the need for a fallback `// 0` guard. Both are stryke built-ins that work in `|>` pipelines. Use `sum0` in contexts where an empty input is expected and you need a safe numeric default.

```
p sum(1:100)    # 5050
p sum0()          # 0
@prices |> map { _->{amount} } |> sum0 |> p
```

product

`product @list` — returns the product of all numeric elements in the list. Returns `undef` for an empty list. Useful for computing factorials, combinatoric products, and compound multipliers. In stryke this is a built-in that composes naturally with `|>` pipelines and `range`.

```
p product(1:5)   # 120
range(1, 10) |> product |> p  # 3628800
```

mean

`mean @list` — returns the arithmetic mean (average) of a numeric list. Computed as `sum / count` in a single pass. Returns `undef` for an empty list. The result is always a floating-point value even if all inputs are integers. Combine with `map` to compute averages over extracted fields.

```
p mean(2, 4, 6, 8)   # 5
@students |> map { _->{score} } |> mean |> p
```

median

`median @list` — returns the median value of a numeric list. For odd-length lists, this is the middle element after sorting. For even-length lists, it is the arithmetic mean of the two middle elements. The input list does not need to be pre-sorted. Returns `undef` for an empty list.

```
p median(1, 3, 5, 7, 9)   # 5
p median(1, 3, 5, 7)      # 4
1:100 |> median |> p     # 50.5
```

mode

`mode @list` — returns the most frequently occurring value in the list. If there is a tie, the value that appears first in the list wins. Returns `undef` for an empty list. Works with both numeric and string values — comparison is done by stringification. Useful for finding the dominant category in a dataset.

```
p mode(1, 2, 2, 3, 3, 3)   # 3
my @logs = qw(INFO WARN INFO ERROR INFO)
p mode(@logs)  # INFO
```

stddev

`stddev @list` (alias `std`) — returns the population standard deviation of a numeric list. This is the square root of the population variance, measuring how spread out values are from the mean. Uses N (not N-1) in the denominator, so it computes the population statistic rather than the sample statistic. Returns `undef` for an empty list.

```
p stddev(2, 4, 4, 4, 5, 5, 7, 9)   # 2
1:10 |> std |> p
```

variance

`variance @list` — returns the population variance of a numeric list, computed as the mean of the squared deviations from the mean. Like `stddev`, this uses N (not N-1) as the divisor. The variance is `stddev ** 2`. Returns `undef` for an empty list. Useful for statistical analysis and as a building block for higher-order stats.

```
p variance(2, 4, 4, 4, 5, 5, 7, 9)   # 4
my @samples = map { rand(100) } 1:1000
p variance(@samples)
```

sample

`sample N, @list` — returns a random sample of N elements drawn without replacement from the list. The returned elements are in random order. If N exceeds the list length, the entire list is returned (shuffled). Uses a Fisher-Yates partial shuffle internally for efficiency. Each call produces a different result due to random selection.

```
my @pick = sample 3, 1:100
@pick |> e p   # 3 random values
my @test_cases = sample 10, @all_cases
```

shuffle

`shuffle @list` — returns a new list with all elements in random order using a Fisher-Yates shuffle. The original list is not modified. Every permutation is equally likely. In stryke this is a global built-in; no import is needed. The alias `shuf` is also available. Commonly used with `|>` and `take` to draw random subsets.

```
my @deck = shuffle 1:52
@deck |> take 5 |> e p   # 5 random cards
my @randomized = shuffle @questions
```

uniq

`uniq @list` — removes duplicate elements from a list, preserving the order of first occurrence. Comparison is done by string equality. The alias `uq` is also available. This is eager (not streaming) and returns a new list. For streaming deduplication in `|>` pipelines, use `distinct`. For type-specific comparison, see `uniqnum`, `uniqstr`, and `uniqint`.

```
my @u = uniq 1, 2, 2, 3, 1, 3
@u |> e p   # 1 2 3
my @hosts = uniq @all_hosts
```

uniqint

`uniqint @list` — removes duplicate elements comparing values as integers. Each element is truncated to its integer part before comparison, so `1.1` and `1.9` are considered equal (both become `1`). The first occurrence is kept. This is useful when you have floating-point data but care only about the integer portion for uniqueness.

```
my @u = uniqint 1, 1.1, 1.9, 2
@u |> e p   # 1 2
my @distinct_ids = uniqint @raw_ids
```

uniqnum

`uniqnum @list` — removes duplicate elements comparing values as floating-point numbers. Unlike `uniq` (which compares as strings), `uniqnum` treats `1.0` and `1.00` as equal because they have the same numeric value. The first occurrence of each numeric value is preserved. Use this when your data contains numbers that may have different string representations.

```
my @u = uniqnum 1.0, 1.00, 2.5, 2.50
@u |> e p   # 1 2.5
my @prices = uniqnum @all_prices
```

uniqstr

`uniqstr @list` — removes duplicate elements comparing values strictly as strings. This is the same comparison as `uniq` but makes the intent explicit. Numeric values `1` and `1.0` are considered different because their string representations differ. Use `uniqstr` when you want to be explicit that string semantics are intended.

```
my @u = uniqstr "a", "b", "a", "c"
@u |> e p   # a b c
my @tags = uniqstr @all_tags
```

zip

`zip(\@a, \@b, ...)` — combines multiple arrays element-wise into a list of arrayrefs. Each output arrayref contains one element from each input array at the corresponding index. Accepts two or more array references. The alias `zp` is also available. By default, `zip` pads shorter arrays with `undef` (equivalent to `zip_longest`). For truncating behavior, use `zip_shortest`.

```
my @a = 1:3
my @b = ("a","b","c")
zip(\@a, \@b) |> e p   # [1,a] [2,b] [3,c]
my @matrix = zip(\@xs, \@ys, \@zs)
```

zip_longest

`zip_longest(\@a, \@b, ...)` — combines arrays element-wise, padding shorter arrays with `undef` to match the longest. This is the explicit version of the default `zip` behavior. Every input array contributes exactly one element per output tuple; missing elements become `undef`. Useful when you need to process all data from unequal-length sources.

```
my @a = 1:3
my @b = ("x")
zip_longest(\@a, \@b) |> e p   # [1,x] [2,undef] [3,undef]
```

zip_shortest

`zip_shortest(\@a, \@b, ...)` — combines arrays element-wise, stopping at the shortest input array. No `undef` padding is produced; the output length equals the minimum input length. Use this when you only want complete tuples and extra trailing elements should be discarded.

```
my @a = 1:5
my @b = ("x","y")
zip_shortest(\@a, \@b) |> e p   # [1,x] [2,y]
my @paired = zip_shortest(\@keys, \@values)
```

chunked

`chunked N, @list` — splits a list into non-overlapping chunks of N elements each. Each chunk is an arrayref. The final chunk may contain fewer than N elements if the list length is not evenly divisible. The alias `chk` is also available. In stryke, `chunked` is eager and returns a list of arrayrefs; for streaming chunk behavior in pipelines, use `chunk`.

```
my @ch = chunked 3, 1:7
@ch |> e p   # [1,2,3] [4,5,6] [7]
1:12 |> chunked 4 |> e { p join ",", @_ }
```

windowed

`windowed N, @list` — returns a sliding window of N consecutive elements over the list. Each window is an arrayref. The output contains `len - N + 1` windows. Unlike `chunked`, windows overlap: each successive window advances by one element. The alias `win` is also available. Useful for computing moving averages, detecting patterns in sequences, or n-gram extraction.

```
my @w = windowed 3, 1:5
@w |> e p   # [1,2,3] [2,3,4] [3,4,5]
my @deltas = windowed(2, @vals) |> map { _->[1] - _->[0] }
```

pairs

`pairs @list` — takes a flat list and groups consecutive elements into pairs, returning a list of two-element arrayrefs (`[$k, $v], ...`). The input list must have an even number of elements. Each pair can be accessed via array indexing (`_->[0]`, `_->[1]`). This is the inverse of `unpairs` and is commonly used to iterate over hash-like flat lists in a structured way.

```
my @p = pairs "a", 1, "b", 2
@p |> e { p "_->[0]=_->[1]" }   # a=1 b=2
my @entries = pairs %hash
```

unpairs

`unpairs @list_of_pairs` — flattens a list of two-element arrayrefs back into a flat key-value list. This is the inverse of `pairs`. Each arrayref `[$k, $v]` becomes two consecutive elements in the output. Useful for converting structured pair data back into a format suitable for hash assignment or flat list processing.

```
my @flat = unpairs ["a",1], ["b",2]
@flat |> e p   # a 1 b 2
my %h = unpairs @filtered_pairs
```

pairkeys

`pairkeys @list` — extracts the keys (even-indexed elements) from a flat pairlist. Given a list like `("a", 1, "b", 2, "c", 3)`, returns `("a", "b", "c")`. This is equivalent to `map { _->[0] } pairs @list` but more concise and efficient. Useful for extracting just the key side of a key-value flat list without constructing intermediate pair objects.

```
my @k = pairkeys "a", 1, "b", 2, "c", 3
@k |> e p   # a b c
my @config_pairs = (host => "localhost", port => 8080)
my @names = pairkeys @config_pairs
```

pairvalues

`pairvalues @list` — extracts the values (odd-indexed elements) from a flat pairlist. Given `("a", 1, "b", 2)`, returns `(1, 2)`. This is equivalent to `map { _->[1] } pairs @list` but more concise. Pair it with `pairkeys` to split a flat key-value list into separate key and value arrays.

```
my @v = pairvalues "a", 1, "b", 2
@v |> e p   # 1 2
my @defaults = (timeout => 30, retries => 3)
my @settings = pairvalues @defaults
```

pairmap

`pairmap BLOCK @list` — maps over consecutive pairs in a flat list, passing the key as `$_0` (or `$a`) and the value as `$_1` (or `$b`). The block can return any number of elements. This is the pair-aware equivalent of `map` and is ideal for transforming hash-like flat lists. The result is a flat list of whatever the block returns.

```
my @out = pairmap { "$_0=$_1" } "a", 1, "b", 2
@out |> e p   # a=1 b=2
my @cfg_pairs = (host => "x", port => 80)
my @upper = pairmap { uc($_0), $_1 } @cfg_pairs
```

pairgrep

`pairgrep { BLOCK } @list` — filters consecutive pairs from a flat list, keeping only those where the block returns true. The key is available as `$_0` (or `$a`) and the value as `$_1` (or `$b`). Returns a flat list of the matching key-value pairs. This is the pair-aware equivalent of `grep` and is useful for filtering hash-like data by both key and value simultaneously.

```
my @big = pairgrep { $_1 > 5 } "a", 3, "b", 9, "c", 1
@big |> e p   # b 9
my @alert_pairs = (info => 1, critical_a => 9, critical_b => 7)
my @important = pairgrep { $_0 =~ /^critical/ } @alert_pairs
```

pairfirst

`pairfirst { BLOCK } @list` — returns the first pair from a flat list where the block returns true, as a two-element list `($key, $value)`. The key is `$_0` (or `$a`) and the value is `$_1` (or `$b`). Short-circuits on the first match. Returns an empty list if no pair matches. This is the pair-aware equivalent of `first`.

```
my @hit = pairfirst { $_1 > 5 } "x", 2, "y", 8
p "@hit"   # y 8
my @flags = (info => 1, debug => 7, trace => 0)
my ($k, $v) = pairfirst { $_0 eq "debug" } @flags
```

mesh

`mesh(\@a, \@b, ...)` — interleaves multiple arrays into a flat list rather than arrayrefs. While `zip` returns `([1,"a"], [2,"b"])`, `mesh` returns `(1, "a", 2, "b")`. This makes it ideal for constructing hashes from parallel key and value arrays. The result is a flat list suitable for direct hash assignment.

```
my @k = ("a","b")
my @v = (1,2)
my %h = mesh(\@k, \@v)
p $h{a}   # 1
my %lookup = mesh(\@ids, \@names)
```

mesh_longest

`mesh_longest(\@a, \@b, ...)` — interleaves arrays into a flat list, padding shorter arrays with `undef` to match the longest. Like `mesh`, the output is flat (not arrayrefs). Missing elements become `undef` in the output sequence. Use this when building a flat interleaved list from arrays of unequal length where you need every element represented.

```
my @a = 1:3
my @b = ("x")
my @r = mesh_longest(\@a, \@b)
@r |> e p   # 1 x 2 undef 3 undef
```

mesh_shortest

`mesh_shortest(\@a, \@b, ...)` — interleaves arrays into a flat list, stopping at the shortest input array. No `undef` padding is produced; trailing elements from longer arrays are silently dropped. Use this when you only want complete interleaved groups and partial data should be discarded.

```
my @a = 1:3
my @b = ("x","y")
my @r = mesh_shortest(\@a, \@b)
@r |> e p   # 1 x 2 y
```

partition

`partition` — functional / iterator builtin. Alias for `take_while`.

```
my $result = partition $x
# or in a pipeline:
@list |> map partition |> p
```

frequencies

`frequencies` (aliases `freq`, `frq`) counts how many times each distinct element appears in a list and returns a hashref mapping each value to its count. This is the stryke equivalent of a histogram or counter — useful for analyzing log files, counting word occurrences, tallying categorical data, or finding duplicates. The input list is flattened, so you can pass arrays directly. The returned hashref can be fed into `dd` for inspection or `to_json` for serialization.

```
my @words = qw(apple banana apple cherry banana apple)
my $counts = freq @words
p $counts   # {apple => 3, banana => 2, cherry => 1}
rl("access.log") |> map { /^(\S+)/ && $1 } |> freq |> dd
my @rolls = map { 1 + int(rand 6) } 1:1000
frq(@rolls) |> to_json |> p
```

tally

`tally` counts how many times each distinct element appears in a list and returns a hash ref mapping element → count. This is the same as Ruby's `Enumerable#tally` or Python's `Counter`. Similar to `frequencies` but follows the Ruby naming convention.

```
my $t = tally("a", "b", "a", "c", "a", "b")
p $t->{a}   # 3
p $t->{b}   # 2
qw(red blue red green blue red) |> tally |> dd
```

interleave

`interleave` (alias `il`) merges two or more arrays by alternating their elements: first element of each array, then second element of each, and so on. If the arrays have different lengths, shorter arrays contribute `undef` for their missing positions. This is useful for building key-value pair lists from separate key and value arrays, creating round-robin schedules, or weaving parallel data streams together.

```
my @keys = qw(name age city)
my @vals = ("Alice", 30, "NYC")
my @pairs = il \@keys, \@vals
p @pairs   # name, Alice, age, 30, city, NYC
my %h = il \@keys, \@vals
p $h{name} # Alice
my @rgb = il [255,0,0], [0,255,0], [0,0,255]
```

pluck

`pluck KEY, LIST_OF_HASHREFS` — extract a single key from each hashref in the list.

For each element, `pluck` dereferences it as a hashref and returns the value at the given key. Elements where the key is missing yield `undef`. This is a streaming operation: in a pipeline, each hashref is processed and the extracted value is emitted immediately. It is the stryke equivalent of `map { _->{KEY} }` but more readable and optimized internally.

```
@users |> pluck "name" |> e p
my @ids = pluck "id", @records
@rows |> pluck "email" |> distinct |> e p
```

grep_v

`grep_v PATTERN, LIST` — inverse grep: reject elements that match the pattern, keep the rest.

This is the complement of `grep` — any element where the pattern matches is dropped, and non-matching elements pass through. It accepts a regex, a string, or a code block as the pattern. In streaming mode, each element is tested and either forwarded or discarded without buffering. This is a stryke built-in that avoids the awkward `grep { !/pattern/ }` double-negation.

```
@words |> grep_v /^ #/ |> e p          # drop comments
my @clean = grep_v qr/tmp/, @files
stdin |> grep_v /^\s*$/ |> e p        # drop blank lines
```

select_keys

`select_keys` — pipeline / string helpers builtin.

```
my $result = select_keys $x
# or in a pipeline:
@list |> map select_keys |> p
```

clamp

`clamp` (alias `clp`) constrains each value in a list to lie within a specified minimum and maximum range. Values below the minimum are raised to it; values above the maximum are lowered to it; values already in range pass through unchanged. This is essential for sanitizing user input, bounding computed values before display, or enforcing physical constraints in simulations. When given a single scalar, it returns a single clamped value.

```
my @scores = (105, -3, 42, 99, 200)
my @clamped = clp 0, 100, @scores
p @clamped   # 100, 0, 42, 99, 100
my $val = clp 0, 255, $input   # bound to byte range
my @pct = map { clp 0.0, 1.0, _ } @raw_ratios
```

normalize

`normalize` (alias `nrm`) rescales a list of numeric values so that the minimum maps to 0 and the maximum maps to 1, using min-max normalization. This is a standard preprocessing step for machine learning features, data visualization (mapping values to color gradients or bar heights), and statistical analysis. If all values are identical, the result is a list of zeros to avoid division by zero. The output preserves the relative ordering of the input.

```
my @temps = (32, 68, 100, 212)
my @normed = nrm @temps
p @normed   # 0, 0.2, 0.377..., 1
my @pixels = nrm @raw_intensities
@pixels |> map { int(_ * 255) } |> e p  # scale to 0-255
```

compose

`compose` (alias `comp`) creates a right-to-left function composition. Given `compose(\&f, \&g)`, calling the result with `x` computes `f(g(x))`. Chain any number of functions — they apply from right to left (last argument first). This is the standard mathemati-

cal function composition found in Haskell, Clojure, and Ramda. The returned code ref can be stored, passed around, or used in pipelines.

```
my $double = fn { $_[0] * 2 }
my $inc    = fn { $_[0] + 1 }
my $f = compose($inc, $double)
p $f->(5)   # 11  (double 5 → 10, inc 10 → 11)

my $pipeline = compose(
    fn { join ",", @{$_[0]} },
    fn { [sort @{$_[0]}] },
    fn { [grep { _ > 2 } @{$_[0]}] },
)
p $pipeline->([3,1,4,1,5])  # 3,4,5
```

partial

`partial` returns a partially applied function — the bound arguments are prepended to any arguments supplied at call time. `partial(\&f, @bound)->(x)` is equivalent to `f(@bound, x)`. This is the standard partial application from functional programming, useful for creating specialized versions of general functions without closures.

```
my $add = fn { $_[0] + $_[1] }
my $add5 = partial($add, 5)
p $add5->(3)   # 8

my $log = fn { p "[$_[0]] $_[1]" }
my $warn_log = partial($log, "WARN")
$warn_log->("disk full")   # [WARN] disk full
```

curry

`curry` auto-curries a function with a given arity. The curried function accumulates arguments across calls and invokes the original only when enough have been collected. `curry(\&f, N)->(a)->(b)` calls `f(a, b)` when N=2. If all arguments are supplied at once, it calls immediately.

```
my $add = curry(fn { $_[0] + $_[1] }, 2)
my $add5 = $add->(5)
p $add5->(3)       # 8
p $add->(10, 20)   # 30  (enough args — calls immediately)
```

memoize

`memoize` (alias `memo`) wraps a function so that repeated calls with the same arguments return a cached result instead of re-executing the function. Arguments are stringified and joined as the cache key. This is essential for expensive pure functions like recursive algorithms, API lookups with stable results, or any computation where the same inputs always produce the same output.

```
my $fib = memoize(fn {
    my $n = $_[0]
    $n < 2 ? $n : $fib->($n-1) + $fib->($n-2)
})
p $fib->(30)   # instant (without memoize: ~1B calls)

my $fetch_user = memo(fn { fetch_json("https://api/users/$_[0]") })
$fetch_user->(42)   # hits API
$fetch_user->(42)   # returns cached
```

once

`once` wraps a function so it is called at most once. The first invocation executes the function and caches the result; all subsequent calls return the cached value without re-executing. This is ideal for lazy initialization, one-time setup, or singleton patterns.

```
my $init = once(fn { p "initializing..."
    42 })
p $init->()   # prints "initializing..." → 42
p $init->()   # 42 (no print — cached)
p $init->()   # 42 (still cached)
```

constantly

`constantly` (alias `const`) returns a function that ignores all arguments and always returns the given value. Useful as a default callback, a stub in higher-order function pipelines, or anywhere a function is required but a fixed value suffices.

```
my $zero = constantly(0)
p $zero->("anything")   # 0
my @defaults = map { constantly(0)->() } 1:5   # [0,0,0,0,0]
```

complement

`complement` (alias `compl`) wraps a predicate function and returns a new function that negates its boolean result. `complement(\&even?)->(3)` returns true. This is the functional equivalent of `!f(x)` without creating a closure.

```
my $even = fn { $_[0] % 2 == 0 }
my $odd = complement($even)
p $odd->(3)   # 1
p $odd->(4)   # 0
1:10 |> grep { complement($even)->(_) } |> e p  # 1 3 5 7 9
```

juxt

`juxt` (juxtapose) takes multiple functions and returns a new function that calls each one with the same arguments and collects the results into an array. This is useful for computing multiple derived values from the same input in a single pass.

```
my $stats = juxt(fn { min @_ }, fn { max @_ }, fn { avg @_ })
my @r = $stats->(3, 1, 4, 1, 5)
p "@r"   # 1 5 2.8
```

fnil

`fnil` wraps a function so that any `undef` arguments are replaced with the given defaults before the function is called. This eliminates repetitive `// $default` patterns inside function bodies.

```
my $greet = fnil(fn { "Hello, $_[0]!" }, "World")
p $greet->(undef)    # Hello, World!
p $greet->("Alice")  # Hello, Alice!
```

deep_clone

`deep_clone` (alias `dclone`) performs a recursive deep copy of a nested data structure. Array refs, hash refs, and scalar refs are cloned recursively so that the result shares no references with the original. Modifications to the clone never affect the source. This is the stryke equivalent of JavaScript's `structuredClone` or Perl's `Storable::dclone`.

```
my $orig = {users => [{name => "Alice"}], meta => {v => 1}}
my $copy = deep_clone($orig)
$copy->{users}[0]{name} = "Bob"
p $orig->{users}[0]{name}   # Alice (unchanged)
```

deep_merge

`deep_merge` (alias `dmerge`) recursively merges two hash references. When both sides have a hash ref for the same key, they are merged recursively; otherwise the right-hand value wins. This is the standard deep merge from Lodash, Ruby's `deep_merge`, and config-file overlay patterns. Returns a new hash ref — neither input is modified.

```
my $defaults = {db => {host => "localhost", port => 5432}, debug => 0}
my $overrides = {db => {port => 3306}, debug => 1}
my $cfg = deep_merge($defaults, $overrides)
p $cfg->{db}{host}   # localhost (from defaults)
p $cfg->{db}{port}   # 3306 (overridden)
p $cfg->{debug}      # 1 (overridden)
```

deep_equal

`deep_equal` (alias `deq`) performs structural equality comparison of two values, recursively descending into array refs, hash refs, and scalar refs. Returns 1 if the structures are identical, 0 otherwise. This is the stryke equivalent of Node's `assert.deepStrictEqual`, Lodash `isEqual`, or Python's `==` on nested dicts/lists.

```
p deep_equal([1, {a => 2}], [1, {a => 2}])   # 1
p deep_equal([1, {a => 2}], [1, {a => 3}])   # 0
p deq({x => [1,2]}, {x => [1,2]})            # 1
```

stryke Extensions

19 topics

fn

Define a named or anonymous function with optional typed parameters and default values. Named functions are installed into the current package; anonymous functions (closures) capture their enclosing lexical scope. Functions are first-class values — assign them to scalars, store in arrays, pass as callbacks. The last expression evaluated is the implicit return value unless an explicit `return` is used.

```
fn dbl($x) { $x * 2 }
my $f = fn { _ * 2 }
my $add = fn ($a: Int, $b: Int) { $a + $b }

# Default parameter values:
fn greet($name = "world") { p "hello $name" }
greet()           # hello world
greet("Alice")    # hello Alice

fn range_check($x, $min = 0, $max = 100) { $min <= $x <= $max }
fn with_list(@items = (1, 2, 3)) { join "-", @items }
fn with_hash(%opts = (debug => 0)) { $opts{debug} }
```

Default values are evaluated at call time if the argument is not provided.

struct

`struct` declares a named record type with typed fields, giving stryke lightweight struct semantics similar to Rust structs or Python dataclasses. Structs support multiple construction syntaxes, default values, field mutation, user-defined methods, functional updates, and structural equality.

Declaration:

```
struct Point { x => Float, y => Float }           # typed fields
struct Point { x => Float = 0.0, y => Float = 0.0 } # with defaults
struct Pair { key, value }                        # untyped (Any)
```

Construction:

```
my $p = Point(x => 1.5, y => 2.0)  # function-call with named args
my $p = Point(1.5, 2.0)            # positional (declaration order)
my $p = Point->new(x => 1.5, y => 2.0) # traditional OO style
my $p = Point()                    # uses defaults if defined
```

Field access (getter/setter):

```
p $p->x       # getter (0 args)
$p->x(3.0)    # setter (1 arg)
```

User-defined methods:

```
struct Circle {
    radius => Float,
    fn area { 3.14159 * $self->radius ** 2 }
    fn scale($factor: Float) {
        Circle(radius => $self->radius * $factor)
    }
}
my $c = Circle(radius => 5)
p $c->area        # 78.53975
p $c->scale(2)    # Circle(radius => 10)
```

Built-in methods:

```
my $q = $p->with(x => 5)  # functional update — new instance
my $h = $p->to_hash       # { x => 1.5, y => 2.0 }
my @f = $p->fields        # (x, y)
my $c = $p->clone         # deep copy
```

Smart stringify:

```
p $p  # Point(x => 1.5, y => 2)
```

Structural equality:

```
my $a = Point(1, 2)
my $b = Point(1, 2)
p $a == $b  # 1 (compares all fields)
```

Note: field type is checked at construction and mutation; unknown field names are fatal errors.

typed

`typed` adds optional runtime type annotations to lexical variables and subroutine parameters. When a `typed` declaration is in effect, stryke inserts a lightweight check at assignment time that verifies the value matches the declared type (`Int`, `Str`, `Float`, `Bool`, `ArrayRef`, `HashRef`, or a user-defined `struct` name). This is especially useful for catching accidental type mismatches at function boundaries in larger programs. The annotation is purely a runtime guard — it has zero impact on pipeline performance because the check is only performed once at the point of assignment, not on every read.

```
typed my $x : Int = 42
typed my $name : Str = "hello"
typed my $pi : Float = 3.14
my $add = fn ($a: Int, $b: Int) { $a + $b }
p $add->(3, 4)   # 7
```

Note: assigning a value of the wrong type raises a runtime exception immediately.

You can mix typed and untyped variables freely in the same scope, so adopting `typed` is incremental — annotate the variables that matter and leave the rest dynamic. Subroutine parameters declared with type annotations in `fn` are checked on every call, giving you contract-style validation at function boundaries without a separate assertion library.

```
typed my @nums : Int = (1, 2, 3)
typed my %cfg : Str = (host => "localhost", port => "8080")
```

match

Algebraic pattern matching (stryke extension).

```
match ($val) {
    /^\d+$/ => p "number: $val",
    [1, 2, _] => p "array starting with 1,2",
    { name => $n } => p "name is $n",
    _ => p "default",
}
```

Patterns: regex, array, hash, literal, wildcard `_`. Optional `if` guard per arm.

frozen

Declare an immutable lexical variable. `const my` and `frozen my` are interchangeable spellings; `const` reads more naturally for engineers coming from other languages.

```
const my $pi = 3.14159
# $pi = 3  # ERROR: cannot assign to frozen variable

frozen my @primes = (2, 3, 5, 7, 11)
```

fore

Side-effect-only list iterator (like `map` but void, returns item count).

```
qw(a b c) |> e p           # prints a, b, c; returns 3
1:5 |> map _ * 2 |> e p  # prints 2,4,6,8,10
```

ep

`ep` — shorthand for `e { p }` (foreach + print). Iterates the list and prints each element.

```
qw(a b c) |> ep            # prints a, b, c (one per line)
filef |> sorted |> ep      # print sorted file list
1:5 |> map _ * 2 |> ep   # prints 2,4,6,8,10
```

p

`p` — alias for `say` (print with newline).

```
p "hello"       # hello\n
p 42            # 42\n
1:5 |> e p     # prints each on its own line
```

gen

Create a generator — lazy `yield` values on demand.

```
my $g = gen { yield _ for 1:5 }
my ($val, $more) = @{$g->next}
```

yield

Yield a value from inside a `gen { }` generator block, suspending the generator until the consumer calls `->next` again. Each `yield` produces one element in the lazy sequence. When the block finishes without yielding, the generator signals exhaustion. This is the stryke equivalent of Python's `yield` or Rust's `Iterator::next`.

```
my $fib = gen {
    my ($a, $b) = (0, 1)
    while (1) {
        yield $a
        ($a, $b) = ($b, $a + $b)
    }
}
for (1:10) {
    my ($val) = @{$fib->next}
    p $val
}
```

trace

Trace `mysync` mutations to stderr (tagged with worker index under `fan`).

```
trace { fan 10 { $counter++ } }
```

timer

Measure wall-clock milliseconds for a block.

```
my $ms = timer { heavy_work() }
```

bench

Benchmark a block N times; returns `"min/mean/p99"`.

```
my $report = bench { work() } 1000
```

eval_timeout

Run a block with a wall-clock timeout (seconds).

```
eval_timeout 5 { slow_operation() }
```

retry

Retry a block on failure.

```
retry { http_call() } times => 3, backoff => 'exponential'
```

rate_limit

Limit invocations per time window.

```
rate_limit(10, "1s") { hit_api() }
```

every

Run a block at a fixed interval.

```
every "500ms" { tick() }
```

watch

Watch a single file for changes (non-parallel).

```
watch "/tmp/x", fn { process }
```

capture

Run a command and capture structured output. Returns an object with `->stdout`, `->stderr`, `->exitcode`, and `->failed` methods. When piped to `lines` or `words`, stdout is auto-extracted so you can chain directly.

```
my $r = capture("ls -la")
p $r->stdout, $r->stderr, $r->exitcode
capture("ps aux") |> lines |> map { columns } |> tbl |> p
```

Data & Serialization

45 topics

json_encode

`json_encode` serializes any stryke data structure — hashrefs, arrayrefs, nested combinations, numbers, strings, booleans, and undef — into a compact JSON string. Native bare-name builtin backed by a Rust serializer; no module to import. The output is always valid UTF-8 JSON suitable for writing to files, sending over HTTP, or piping to other tools. Use `json_decode` to round-trip back.

```
my %cfg = (debug => 1, paths => ["/tmp", "/var"])
my $j = json_encode(\%cfg)
p $j   # {"debug":1,"paths":["/tmp","/var"]}
$j |> spurt "/tmp/cfg.json"$1

Note: undef becomes JSON `null`; booleans serialize as `true`/`false`.
```

json_decode

`json_decode` parses a JSON string and returns the corresponding Perl data structure — hashrefs for objects, arrayrefs for arrays, and native scalars for strings/numbers/booleans. It is strict by default: malformed JSON raises an exception rather than returning partial data. This makes it safe to use in pipelines where corrupt input should halt processing. The Rust parser underneath handles large documents efficiently and supports full Unicode.

```
my $data = json_decode('{"name":"stryke","ver":1}')
p $data->{name}   # stryke
slurp("data.json") |> json_decode |> dd$1

Note: JSON `null` becomes Perl `undef`; trailing commas and comments are not allowed.
```

json_jq

`json_jq` applies a jq-style query expression to a stryke data structure and returns the matched value. This brings the power of the `jq` command-line JSON processor directly into stryke without shelling out. Dot notation traverses hash keys, bracket notation indexes arrays, and nested paths are separated by dots. It is ideal for extracting deeply nested values from API responses or config files without chains of hash dereferences.

```
my $data = json_decode(slurp "api.json")
p json_jq($data, ".results[0].name")
my $cfg = rj "config.json"
p json_jq($cfg, ".database.host")  # deep extract
fetch_json("https://api.example.com/users") |> json_jq(".data[0].email") |> p
```

to_json

`to_json` (alias `tj`) converts a stryke data structure into a JSON string, functioning as a convenient shorthand for `json_encode`. It accepts hashrefs, arrayrefs, scalars, and nested combinations, producing compact JSON output suitable for APIs, config files, or inter-process communication. The alias `tj` is particularly useful at the end of a pipeline to serialize the final result. The Rust-backed serializer handles large structures efficiently and always produces valid UTF-8.

```
my %user = (name => "Bob", age => 30)
p tj \%user   # {"age":30,"name":"Bob"}
my @items = map { {id => _, val => _ * 2} } 1:3
p tj \@items  # [{"id":1,"val":2},{"id":2,"val":4},{"id":3,"val":6}]
@data |> tj |> spurt "out.json"
```

to_csv

to_csv (alias tc) serializes a list of hashrefs or arrayrefs into a CSV-formatted string, complete with a header row derived from hash keys when given hashrefs. This is the fastest way to produce spreadsheet-ready output from structured data. Fields containing commas, quotes, or newlines are automatically escaped according to RFC 4180. The alias tc keeps one-liners terse when piping query results or API responses straight to CSV.

```
my @rows = ({name => "Alice", age => 30}, {name => "Bob", age => 25})
p tc \@rows   # name,age\nAlice,30\nBob,25
tc(\@rows) |> spurt "people.csv"
my @grid = ([1, 2, 3], [4, 5, 6])
p tc \@grid   # 1,2,3\n4,5,6
```

to_toml

to_toml (alias tt) serializes a stryke hashref into a TOML-formatted string. TOML is a popular configuration format that maps cleanly to hash structures, making tt ideal for generating config files programmatically. Nested hashes become TOML sections, arrays become TOML arrays, and scalar values are serialized with their natural types. The output is always valid TOML that can be parsed back with toml_decode.

```
my %cfg = (database => {host => "localhost", port => 5432}, debug => 1)
p tt \%cfg
# [database]
# host = "localhost"
# port = 5432
# debug = 1
tt(\%cfg) |> spurt "config.toml"
```

to_yaml

to_yaml (alias ty) serializes a stryke data structure into a YAML-formatted string. YAML is widely used for configuration and data exchange where human readability matters. Nested structures are represented with indentation, arrays with leading dashes, and strings are quoted only when necessary. The output is valid YAML 1.2 that round-trips cleanly through yaml_decode. The alias ty is convenient for quick inspection of complex data.

```
my %app = (name => "myapp", deps => ["tokio", "serde"], version => "1.0")
p ty \%app
# name: myapp
# version: "1.0"
# deps:
#   - tokio
#   - serde
ty(\%app) |> spurt "app.yaml"
```

to_xml

to_xml (alias tx) serializes a stryke data structure into an XML string. Hash keys become element names, array values become repeated child elements, and scalar values become text content. This is useful for generating XML payloads for SOAP APIs, RSS feeds, or configuration files that require XML format. The output is well-formed XML that can be parsed back with xml_decode.

```
my %doc = (root => {title => "Hello", items => ["a", "b", "c"]})
p tx \%doc
# <root><title>Hello</title><items>a</items><items>b</items><items>c</items></root>
tx(\%doc) |> spurt "doc.xml"
```

to_html

to_html (alias th) serializes a stryke data structure into a self-contained HTML document with cyberpunk styling (dark background, neon cyan/magenta accents, monospace fonts). Arrays of hashrefs render as full tables with headers, plain arrays as bullet lists, single hashes as key-value tables, and scalars as styled text blocks. Pipe to a file and open in a browser for instant data visualization.

```
fr |> map +{name => _, size => format_bytes(size)} |> th |> to_file("report.html")
my @rows = ({name => "Alice", age => 30}, {name => "Bob", age => 25})
@rows |> th |> to_file("people.html")
th({host => "localhost", port => 5432}) |> p
```

to_markdown

`to_markdown` (aliases `to_md`, `tmd`) serializes a stryke data structure into Markdown text. Arrays of hashrefs render as GFM tables with headers and separator rows, plain arrays as bullet lists, single hashes as 2-column key-value tables, and scalars as plain text. The output is valid GitHub-Flavored Markdown suitable for README files, issue comments, or any Markdown renderer.

```
fr |> map +{name => _, size => format_bytes(size)} |> tmd |> to_file("report.md")
my @rows = ({name => "Alice", age => 30}, {name => "Bob", age => 25})
@rows |> tmd |> p
# | name | age |
# | --- | --- |
# | Alice | 30 |
# | Bob | 25 |
```

from_json

`from_json` (alias `fj`) parses a JSON string into a stryke data structure (hashref, arrayref, scalar). This is the inverse of `to_json` and the idiomatic way to decode JSON responses from APIs, config files, or inter-process communication. The Rust-backed parser handles large documents efficiently.

```
my $json = '{"name":"Alice","age":30}'
my $data = from_json($json)
p $data->{name}   # Alice
my $arr = fj '[1, 2, {"x": 3}]'
p $arr->[2]{x}    # 3
slurp("config.json") |> fj
```

from_yaml

`from_yaml` (alias `fy`) parses a YAML string into a stryke data structure. YAML is commonly used for configuration files due to its human-readable syntax. Handles scalars, arrays, hashes, and nested combinations. The parser supports common YAML features like multi-line strings and bare unquoted keys.

```
my $yaml = "name: Alice\nage: 30"
my $data = from_yaml($yaml)
p $data->{name}   # Alice
slurp("config.yaml") |> fy |> say _->{database}{host}
```

from_toml

`from_toml` (alias `ftoml`) parses a TOML string into a stryke hashref. TOML is popular for Rust, Python, and other config files due to its explicit syntax. Sections become nested hashes, arrays stay arrays, and typed values (integers, floats, strings, booleans) are preserved.

```
my $toml = "[package]\nname = \"myapp\"\nversion = \"1.0\""
my $data = from_toml($toml)
p $data->{package}{name}   # myapp
slurp("Cargo.toml") |> ftoml |> say _->{package}{version}
```

from_xml

`from_xml` (alias `fx`) parses an XML string into a stryke hashref. Element names become hash keys, text content becomes string values, nested elements become nested hashes. The parser handles the XML declaration and basic structures.

```
my $xml = '<root><name>Alice</name><age>30</age></root>'
my $data = from_xml($xml)
p $data->{root}{name}   # Alice
slurp("config.xml") |> fx |> say _->{config}{database}{host}
```

from_csv

`from_csv` (alias `fcsv`) parses a CSV string into an arrayref of hashrefs, treating the first line as headers. This is the inverse of `to_csv` and handles quoted fields containing commas according to RFC 4180.

```
my $csv = "name,age\nAlice,30\nBob,25"
my $rows = from_csv($csv)
p $rows->[0]{name}   # Alice
p $rows->[1]{age}    # 25
slurp("data.csv") |> fcsv |> grep { _->{age} > 25 }
```

xopen

`xopen` (alias `xo`) opens a file or URL with the system default handler — `open` on macOS, `xdg-open` on Linux, `start` on Windows. Returns the path unchanged so it can sit transparently in a pipeline. This is the missing link between generating output files and viewing them: pipe the path through `xopen` and the OS opens it in the appropriate app (browser for HTML, viewer for PDF, editor for text).

```
fr |> map +{name => _, size => format_bytes(size)} |> th |> to_file("report.html") |> xopen
xopen "https://example.com"
xopen "output.pdf"
```

clip

`clip` (aliases `clipboard`, `pbcopy`) copies text to the system clipboard (`pbcopy` on macOS, `xclip`/`xsel` on Linux). Returns the text unchanged for pipeline chaining. This is the missing link between generating output and sharing it — pipe any serializer's output through `clip` and paste directly into Slack, GitHub, docs, or email.

```
fr |> map +{name => _, size => format_bytes(size)} |> tmd |> clip   # markdown table → clipboard
qw(a b c) |> join "," |> clip |> p                                  # also prints
"some text" |> clip                                                  # just copy
```

paste

`paste` (alias `pbpaste`) reads text from the system clipboard (`pbpaste` on macOS, `xclip`/`xsel` on Linux). Returns the clipboard contents as a string, ready for pipeline processing.

```
p paste                              # print clipboard contents
paste |> lines |> grep /error/i |> p  # search clipboard for errors
paste |> wc |> p                      # word count of clipboard
```

to_table

`to_table` (aliases `table`, `tbl`) renders data as a plain-text column-aligned table with Unicode box-drawing borders. Arrays of hashrefs render as full tables with headers, plain arrays as numbered rows, single hashes as key-value tables sorted by numeric value descending (ties by key), matching `freq` | `tbl` expectations. The output is fixed-width text suitable for terminal display, log files, or any monospace context.

```
my @r = ({name => "Alice", age => 30}, {name => "Bob", age => 25})
@r |> tbl |> p
# ┌───────┬─────┐
# │ name  │ age │
# ├───────┼─────┤
# │ Alice │ 30  │
# │ Bob   │ 25  │
# └───────┴─────┘
fr |> map +{name => _, size => format_bytes(size)} |> tbl |> p
```

sparkline

`sparkline` (alias `spark`) renders a list of numbers as a compact Unicode sparkline string using block characters (▁▂▃▄▅▆▇█). Each value maps to a bar height proportional to the min/max range. Ideal for inline data visualization in terminal output, dashboards, or log summaries.

```
(3,7,1,9,4,2,8,5) |> spark |> p   # ▃▆▁▇▄▂▆▃
@daily_sales |> spark |> p         # quick trend line
fr |> map { size } |> spark |> p   # file size distribution
```

bar_chart

`bar_chart` (alias `bars`) renders a hashref as a colored horizontal bar chart in the terminal. Each key becomes a label, each value a proportional bar. Colors cycle automatically. Pairs naturally with `freq` for instant word/event counting visualization.

```
qw(a b a c a b) |> freq |> bars |> p
# a ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 3
# b ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 2
# c ▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 1
bars({cpu => 73, mem => 45, disk => 91}) |> p
```

flame

`flame` (alias `flamechart`) renders a hierarchical hashref as a terminal flamechart with colored stacked bars. Nested hashes become child rows; widths are proportional to leaf values (weights). Useful for visualizing call stacks, cost breakdowns, or any tree-structured data.

```
flame({main => {parse => 30, eval => {compile => 15, run => 45}}, init => 10}) |> p
my $profile = read_json "profile.json"
flame($profile) |> p
```

histo

`histo` renders a vertical histogram in the terminal. Given a hashref (label → count), it draws colored vertical bars. Given a flat array of numbers, it auto-bins into 10 buckets and shows the distribution. Pairs with `freq` for categorical data.

```
qw(a b a c a b) |> freq |> histo |> p   # vertical bars for a/b/c
(map { int(rand(100)) } 1:1000) |> histo |> p  # distribution of randoms
```

gauge

`gauge` renders a single value as a horizontal gauge bar with color coding (green ≥80%, yellow ≥50%, magenta ≥25%, red below). Accepts a fraction (0.0–1.0) or a value with max: `gauge(45, 100)`.

```
p gauge(0.73)       # [▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇] 73%
p gauge(45, 100)    # [▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇] 45%
```

spinner

`spinner` shows an animated braille spinner on stderr while a block executes, then clears the spinner and returns the block's result. Useful for long-running one-liners where you want visual feedback without polluting stdout.

```
my $r = spinner "loading" { fetch_json("https://api.example.com/data") }
spinner { sleep 2
    42 }   # default message "working"
```

csv_read

`csv_read` (alias `cr`) reads CSV data from a file path or string and returns an array of arrayrefs, where each inner arrayref represents one row. The parser handles RFC 4180 CSV correctly — quoted fields with embedded commas, newlines inside quotes, and escaped double quotes are all supported. The first row is treated as data (not a header) unless you process it separately. This is the fastest way to ingest tabular data in stryke.

```
my @rows = csv_read "data.csv"
p $rows[0]            # first row as arrayref
@rows |> e { p _->[0] }  # print first column
my @inline = csv_read "a,b,c\n1,2,3\n4,5,6"
p scalar @inline       # 3 rows (including header)
```

csv_write

`csv_write` (alias `cw`) serializes an array of arrayrefs into a CSV-formatted string. Each inner arrayref becomes one row, and fields are automatically quoted when they contain commas, quotes, or newlines. This is the complement of `csv_read` and produces output conforming to RFC 4180. Use it to generate CSV files from computed data, export database query results, or prepare data for spreadsheet import.

```
my @data = (["name", "age"], ["Alice", 30], ["Bob", 25])
csv_write(\@data) |> spurt "people.csv"
my @report = map { [_->{id}, _->{score}] } @results
p csv_write \@report
```

dataframe

`dataframe` (alias `df`) creates a columnar dataframe from tabular data, providing a structured way to work with rows and columns. You can construct a dataframe from an array of hashrefs, an array of arrayrefs with a header row, or from a CSV file. The dataframe supports column selection, filtering, sorting, and aggregation operations. This is stryke's answer to Python's pandas DataFrame — lightweight but sufficient for common data manipulation tasks.

```
my $df = df [{name => "Alice", age => 30}, {name => "Bob", age => 25}]
p $df
my $df2 = df csv_read "data.csv"
my @ages = $df->{age}
p @ages   # 30, 25
```

sqlite

`sqlite` (alias `sql`) executes a SQL statement against an SQLite database file and returns the results. For SELECT queries, it returns an array of hashrefs where each hashref represents one row with column names as keys. For INSERT, UPDATE, and DELETE statements, it returns the number of affected rows. Bind parameters prevent SQL injection and handle quoting automatically. The database file is created if it does not exist, making `sqlite` a zero-setup embedded database.

```
sqlite("app.db", "CREATE TABLE users (name TEXT, age INT)")
sqlite("app.db", "INSERT INTO users VALUES (?, ?)", "Alice", 30)
my @rows = sqlite("app.db", "SELECT * FROM users WHERE age > ?", 20)
@rows |> e { p _->{name} }
sqlite("app.db", "SELECT count(*) as n FROM users") |> dd
```

digits

`digits` (alias `dg`) extracts all digit characters from a string and returns them as a list. This is a quick way to pull numbers out of mixed text without writing a regex. Pairs naturally with `join` to reconstruct the numeric string, or `freq` for digit distribution analysis.

```
p join "", digits("phone: 555-1234")     # 5551234
"abc123def456" |> digits |> cnt |> p     # 6
cat("log.txt") |> digits |> freq |> bars |> p
```

letters

`letters` (alias `lts`) extracts all alphabetic characters from a string and returns them as a list. Filters out digits, punctuation, whitespace — keeps only letters.

```
p join "", letters("h3ll0 w0rld!")   # hllwrld
"abc123DEF" |> letters |> cnt |> p   # 6
```

sentences

`sentences` (alias `sents`) splits text on sentence boundaries (`. ! ?` followed by whitespace or end of string). Returns a list of trimmed sentences. Useful for NLP pipelines, text analysis, or splitting prose into processable units.

```
"Hello world. How are you? Fine!" |> sentences |> e p
# Hello world.
# How are you?
# Fine!
cat("essay.txt") |> sentences |> cnt |> p    # sentence count
```

paragraphs

`paragraphs` (alias `paras`) splits text on blank lines (one or more consecutive newlines). Returns a list of trimmed paragraph strings. Useful for processing structured documents, README files, or any text with paragraph breaks.

```
cat("README.md") |> paragraphs |> cnt |> p   # paragraph count
cat("essay.txt") |> paragraphs |> map { sentences |> cnt } |> spark |> p  # sentences per paragraph
```

sections

`sections` (alias `sects`) splits text on markdown-style headers (`# ...`, `## ...`) or lines of `===`/`---`. Returns a list of arrayrefs `[heading, body]` where each section's heading and body are separated. Useful for parsing structured documents.

```
cat("README.md") |> sections |> cnt |> p     # section count
cat("README.md") |> sections |> map { _->[0] } |> e p  # list all headings
```

numbers

`numbers` (alias `nums`) extracts all numbers (integers and floats, including negatives) from a string and returns them as numeric values. Unlike `digits` which returns individual digit characters, `numbers` returns actual parsed values. Useful for pulling measurements, scores, or IDs out of mixed text.

```
p join ",", numbers("temp 98.6F, -20C, ver 3")  # 98.6,-20,3
cat("log.txt") |> numbers |> avg |> p              # average of all numbers in file
"price: $12.99 qty: 5" |> numbers |> e p           # 12.99, 5
```

graphemes

`graphemes` (alias `grs`) splits a string into Unicode grapheme clusters. Unlike `chars` which splits on code points, `graphemes` keeps combining marks and emoji sequences together as single visual units. `"cafe\u{0301}"` gives 4 graphemes (not 5 code points). Essential for correct text processing of accented characters, emoji, and non-Latin scripts.

```
p cnt graphemes("cafe\u{0301}")    # 4 (not 5)
graphemes("hello") |> e p          # h, e, l, l, o
```

columns

`columns` (alias `cols`) splits fixed-width columnar text into fields. Without a widths argument, it auto-detects columns by splitting on runs of 2+ whitespace (ideal for `ps aux`, `ls -l`, `df` output). With an arrayref of widths, it splits at exact fixed positions. Pairs with `lines` for processing tabular command output.

```
my @fields = columns("USER  PID  %CPU")            # auto-detect: [USER, PID, %CPU]
my @fixed = columns("John   Doe   30", [7, 7, 3])   # fixed-width: [John, Doe, 30]
capture("ps aux") |> lines |> map { columns } |> tbl |> p
```

stringify

`stringify` (alias `str`) converts any stryke value — scalars, array refs, hash refs, nested structures, undef — into a string representation that is a valid stryke literal. The output is designed for round-tripping: you can `eval` the returned string to reconstruct

the original data structure. This makes it ideal for serializing state to a file in a Perl-native format, generating code fragments, or building reproducible test fixtures. Unlike `dd`, which targets human readability, `str` prioritizes parseability.

```
my $s = str {a => 1, b => [2, 3]}
p $s              # {a => 1, b => [2, 3]}
my $copy = eval $s # round-trip back to hashref
my @list = (1, "hello", undef)
p str \@list        # [1, "hello", undef]
```

Note: references are serialized recursively; circular references will cause infinite recursion.

ddump

`ddump` (alias `dd`) pretty-prints any stryke data structure to stderr in a human-readable, indented format. Native bare-name builtin — no module to import. It is the go-to tool for quick debugging — drop a `dd` call anywhere in a pipeline to inspect intermediate values without disrupting the data flow. The output is colorized when stderr is a terminal. Unlike `str`, the output is not meant for `eval` round-tripping; it prioritizes clarity over parseability. `dd` returns its argument unchanged, so it can be inserted into pipelines transparently.

```
my %h = (name => "Alice", scores => [98, 87, 95])
dd \%h                    # pretty-prints to stderr
my @result = @data |> dd |> grep { _->{active} } |> dd
dd [1, {a => 2}, [3, 4]]     # nested structure
```

Note: `dd` writes to stderr, not stdout, so it never contaminates pipeline output.

toml_decode

`toml_decode` (alias `td`) parses a TOML-formatted string and returns the corresponding stryke hash structure. TOML sections become nested hashrefs, arrays map to arrayrefs, and typed scalars (integers, floats, booleans, datetime strings) are preserved. This is useful for reading configuration files, parsing Cargo.toml manifests, or processing any TOML input. Malformed TOML raises an exception.

```
my $cfg = toml_decode(slurp "config.toml")
p $cfg->{database}{host}   # localhost
my $cargo = toml_decode(slurp "Cargo.toml")
p $cargo->{package}{version}
slurp("settings.toml") |> toml_decode |> dd
```

toml_encode

`toml_encode` (alias `te`) serializes a stryke hashref into a valid TOML string. Nested hashes become TOML sections with `[section]` headers, arrays become TOML arrays, and scalars are serialized with appropriate quoting. This is the inverse of `toml_decode` and is useful for generating or updating configuration files programmatically. The output is human-readable and can be written directly to a `.toml` file.

```
my %cfg = (server => {host => "0.0.0.0", port => 8080}, debug => 0)
toml_encode(\%cfg) |> spurt "server.toml"
my $round = toml_decode(toml_encode(\%cfg))
p $round->{server}{port}  # 8080
```

yaml_decode

`yaml_decode` (alias `yd`) parses a YAML string and returns the corresponding stryke data structure. Mappings become hashrefs, sequences become arrayrefs, and scalars are coerced to their natural Perl types. This handles YAML 1.2 including multi-document streams, anchors/aliases, and flow notation. It is the go-to function for reading YAML configuration files, Kubernetes manifests, or CI pipeline definitions. Invalid YAML raises an exception.

```
my $cfg = yaml_decode(slurp "docker-compose.yml")
p $cfg->{services}{web}{image}
my $ci = yaml_decode(slurp ".github/workflows/ci.yml")
dd $ci->{jobs}
slurp("values.yaml") |> yaml_decode |> json_encode |> p
```

yaml_encode

yaml_encode (alias ye) serializes a stryke data structure into a valid YAML string. Hashes become YAML mappings, arrays become sequences with dash prefixes, and scalars are quoted only when necessary for disambiguation. The output is human-readable and suitable for writing config files, generating Kubernetes resources, or producing YAML-based API payloads. It is the inverse of yaml_decode.

```
my %svc = (name => "api", replicas => 3, ports => [80, 443])
yaml_encode(\%svc) |> spurt "service.yaml"
my $round = yaml_decode(yaml_encode(\%svc))
p $round->{replicas}   # 3
rj("config.json") |> yaml_encode |> p  # JSON to YAML
```

xml_decode

xml_decode (alias xd) parses an XML string and returns a stryke data structure. Elements become hash keys, text content becomes scalar values, repeated child elements become arrayrefs, and attributes are accessible through a conventions-based mapping. This is useful for consuming SOAP responses, RSS feeds, SVG files, or any XML-based API. Malformed XML raises an exception rather than returning partial data.

```
my $doc = xml_decode('<root><name>stryke</name><ver>1</ver></root>')
p $doc->{root}{name}   # stryke
slurp("feed.xml") |> xml_decode |> dd
my $svg = xml_decode(fetch("https://example.com/image.svg"))
p $svg->{svg}
```

xml_encode

xml_encode (alias xe) serializes a stryke data structure into a well-formed XML string. Hash keys become element names, scalar values become text content, and arrayrefs become repeated sibling elements. This is useful for generating XML payloads for SOAP APIs, creating RSS or Atom feeds, or producing configuration files in XML format. The output is the inverse of xml_decode and round-trips cleanly.

```
my %data = (root => {title => "Test", items => [{id => 1}, {id => 2}]})
p xml_encode \%data
xml_encode(\%data) |> spurt "output.xml"
my $payload = xml_encode {request => {action => "query", id => 42}}
http_request(method => "POST", url => $endpoint, body => $payload)
```

HTTP & Networking

23 topics

fetch

`fetch` (alias `ft`) performs a blocking HTTP GET request to the given URL and returns the response body as a string. It follows redirects automatically and raises an exception on network errors or non-2xx status codes. This is the simplest way to retrieve content from the web in stryke — no need to configure a client or parse response objects. For JSON APIs, prefer `fetch_json` which additionally decodes the response.

```
my $html = fetch "https://example.com"
p $html
my $ip = fetch "https://api.ipify.org"
p "My IP: $ip"
fetch("https://example.com/data.txt") |> spurt "local.txt"$1

Note: for POST, PUT, DELETE, or custom headers, use `http_request` instead.
```

fetch_json

`fetch_json` (alias `ftj`) performs a blocking HTTP GET request and automatically decodes the JSON response body into a stryke data structure. This combines `fetch` and `json_decode` into a single call, which is the common case when consuming REST APIs. It raises an exception if the response is not valid JSON or the request fails. The returned value is typically a hashref or arrayref ready for immediate use.

```
my $data = fetch_json "https://api.github.com/repos/stryke/stryke"
p $data->{stargazers_count}
fetch_json("https://jsonplaceholder.typicode.com/todos") |> e { p _->{title} }
my $weather = fetch_json "https://wttr.in/?format=j1"
p $weather->{current_condition}[0]{temp_C}
```

fetch_async

`fetch_async` (alias `fta`) initiates a non-blocking HTTP GET request and returns a task handle that can be awaited later. This allows you to fire off multiple HTTP requests concurrently and wait for them all to complete, dramatically reducing total latency when fetching from several endpoints. The task resolves to the response body as a string, just like `fetch`. Use this when you need to parallelize network I/O without threads.

```
my $t1 = fetch_async "https://api.example.com/users"
my $t2 = fetch_async "https://api.example.com/posts"
my $users = await $t1
my $posts = await $t2
p "Got users and posts concurrently"
```

fetch_async_json

`fetch_async_json` (alias `ftaj`) initiates a non-blocking HTTP GET request that automatically decodes the JSON response when awaited. This combines `fetch_async` with `json_decode`, making it the ideal choice for concurrent API calls where every response is JSON. The task resolves to the decoded stryke data structure — a hashref, arrayref, or scalar depending on the JSON content.

```
my @urls = map { "https://api.example.com/item/$_" } 1:10
my @tasks = map fetch_async_json @urls
my @results = map await @tasks
@results |> e { p _->{name} }
```

http_request

`http_request` (alias `hr`) performs a fully configurable HTTP request with control over method, headers, body, and timeout. Unlike `fetch` which is limited to GET, `http_request` supports POST, PUT, PATCH, DELETE, and any other HTTP method. Pass named parameters for configuration. The response is returned as a hashref containing `status`, `headers`, and `body` fields, giving you full access to the HTTP response. This is the right tool when you need to send data, set authentication headers, or inspect status codes.

```
my $res = http_request(method => "POST", url => "https://api.example.com/users",
    headers => {"Content-Type" => "application/json", Authorization => "Bearer $token"},
    body => tj {name => "Alice"})
p $res->{status}   # 201
my $data = json_decode $res->{body}
my $del = http_request(method => "DELETE", url => "https://api.example.com/users/42")
```

par_fetch

`par_fetch @urls` — fetch a list of URLs in parallel using async HTTP, returning an array of response bodies in input order.

Under the hood, stryke spawns concurrent HTTP GET requests across a connection pool with keep-alive and automatic retry on transient failures (5xx, timeouts). The degree of parallelism is bounded by the connection pool size (default 64) to avoid overwhelming the target server. For heterogeneous HTTP methods or custom headers, use `http_request` inside `pmap` instead. `par_fetch` is the right tool when you have a homogeneous list of URLs and just need the bodies — it handles connection reuse, DNS caching, and TLS session resumption automatically for maximum throughput.

```
my @bodies = par_fetch @urls

# Fetch and decode JSON in one shot:
my @data = par_fetch(@api_urls) |> map json_decode

# Download pages with progress:
my @pages = par_fetch @urls, progress => 1
```

serve

Start a blocking HTTP server.

```
serve 8080, fn ($req) {
    # $req = { method, path, query, headers, body, peer }
    { status => 200, body => "hello" }
}

serve 3000, fn ($req) {
    my $data = { name => "stryke", version => "0.4" }
    { status => 200, body => json_encode($data) }
}, { workers => 8 }$1

Handler returns: hashref `{ status, body, headers }`, string (200 OK), or undef (404).
JSON content-type auto-detected when body starts with `{` or `[`.
```

socket

Create a network socket with the specified domain, type, and protocol. The socket handle is stored in the first argument and can then be used with `bind`, `connect`, `send`, `recv`, and other socket operations. Domain constants include `AF_INET` (IPv4) and `AF_INET6` (IPv6); type constants include `SOCK_STREAM` (TCP) and `SOCK_DGRAM` (UDP).

```
socket(my $sock, AF_INET, SOCK_STREAM, 0)
my $addr = sockaddr_in(8080, inet_aton("127.0.0.1"))
connect($sock, $addr)
send($sock, "GET / HTTP/1.0\r\n\r\n", 0)
```

bind

Bind a socket to a local address so it can accept connections or receive datagrams on that address. The address is a packed `sockaddr_in` or `sockaddr_in6` structure. Binding is required before calling `listen` on a server socket. Dies if the address is already in use unless `SO_REUSEADDR` is set.

```
socket(my $srv, AF_INET, SOCK_STREAM, 0)
setsockopt($srv, SOL_SOCKET, SO_REUSEADDR, 1)
bind($srv, sockaddr_in(8080, INADDR_ANY)) or die "bind: $!"
listen($srv, 5)
```

listen

Mark a bound socket as passive, ready to accept incoming connections. The backlog argument specifies the maximum number of pending connections the OS will queue before refusing new ones. This is only meaningful for stream (TCP) sockets. Call `accept` in a loop after `listen` to handle clients.

```
socket(my $srv, AF_INET, SOCK_STREAM, 0)
bind($srv, sockaddr_in(9000, INADDR_ANY))
listen($srv, 128) or die "listen: $!"
while (accept(my $client, $srv)) {
    send($client, "hello\n", 0)
}
```

accept

Accept a pending connection on a listening socket and return a new connected socket handle. The new handle is used for communication with that specific client while the original listening socket continues accepting others. Returns the packed remote address on success, false on failure.

```
listen($srv, 5)
while (my $remote = accept(my $client, $srv)) {
    my ($port, $ip) = sockaddr_in($remote)
    p "connection from " . inet_ntoa($ip) . ":$port"
    send($client, "welcome\n", 0)
}
```

connect

Initiate a connection from a socket to a remote address. For TCP sockets this performs the three-way handshake; for UDP it sets the default destination so subsequent `send` calls do not need an address. Dies or returns false if the connection is refused or times out.

```
socket(my $sock, AF_INET, SOCK_STREAM, 0)
my $addr = sockaddr_in(80, inet_aton("example.com"))
connect($sock, $addr) or die "connect: $!"
send($sock, "GET / HTTP/1.0\r\n\r\n", 0)
recv($sock, my $buf, 4096, 0)
p $buf
```

send

Send data through a connected socket. The flags argument controls behavior — use `0` for normal sends. For UDP sockets you can supply a destination address as a fourth argument to send to a specific peer without calling `connect` first. Returns the number of bytes sent, or `undef` on error.

```
send($sock, "hello world\n", 0)
my $n = send($sock, $payload, 0)
p "sent $n bytes"
# UDP to specific peer
send($udp, $msg, 0, sockaddr_in(5000, inet_aton("10.0.0.1")))
```

recv

Receive data from a socket into a buffer. The length argument specifies the maximum number of bytes to read. For stream sockets a short read is normal — loop until you have all expected data. For datagram sockets each call returns exactly one datagram. Returns the sender address for UDP, or empty string for TCP.

```
recv($sock, my $buf, 4096, 0) or die "recv: $!"
p $buf
my $data = ""
while (recv($sock, my $chunk, 8192, 0) && length($chunk)) {
    $data .= $chunk
}
```

shutdown

Shut down part or all of a socket connection without closing the file descriptor. The `how` argument controls direction: `0` stops reading, `1` stops writing (sends FIN to peer), `2` stops both. This is useful for signaling end-of-data to the remote side while still reading its response.

```
send($sock, $request, 0)
shutdown($sock, 1)          # done writing
recv($sock, my $resp, 65536, 0)  # still read response
shutdown($sock, 2)          # fully close
```

setsockopt

Set an option on a socket at the specified protocol level. Common uses include enabling `SO_REUSEADDR` to allow immediate rebinding after a server restart, setting `TCP_NODELAY` to disable Nagle's algorithm, or adjusting buffer sizes. The value is typically a packed integer.

```
setsockopt($srv, SOL_SOCKET, SO_REUSEADDR, 1)
setsockopt($sock, IPPROTO_TCP, TCP_NODELAY, 1)
setsockopt($sock, SOL_SOCKET, SO_RCVBUF, pack("I", 262144))
```

getsockopt

Retrieve the current value of a socket option at the specified protocol level. Returns the option value as a packed binary string — use `unpack` to interpret it. Useful for inspecting buffer sizes, checking whether `SO_REUSEADDR` is set, or reading OS-assigned values.

```
my $val = getsockopt($sock, SOL_SOCKET, SO_RCVBUF)
p unpack("I", $val)               # e.g. 131072
my $reuse = getsockopt($srv, SOL_SOCKET, SO_REUSEADDR)
p unpack("I", $reuse)             # 1 or 0
```

getsockname

Return the packed socket address of the local end of a socket. This is useful when the socket was bound to `INADDR_ANY` or port `0` (OS-assigned) and you need to discover the actual address and port the OS chose. Unpack the result with `sockaddr_in`.

```
bind($sock, sockaddr_in(0, INADDR_ANY))  # OS picks port
my $local = getsockname($sock)
my ($port, $ip) = sockaddr_in($local)
p "listening on port $port"
```

getpeername

Return the packed socket address of the remote end of a connected socket. Use `sockaddr_in` or `sockaddr_in6` to unpack it into a port and IP address. This is how a server discovers which client it is talking to after `accept`, or how a client confirms the peer address after `connect`.

```
my $packed = getpeername($client)
my ($port, $ip) = sockaddr_in($packed)
p "peer: " . inet_ntoa($ip) . ":$port"
```

gethostbyname

Resolve a hostname to its network addresses using the system resolver. Returns `($name, $aliases, $addrtype, $length, @addrs)`. The addresses are packed binary — pass them through `inet_ntoa` to get dotted-quad strings. This is the classic DNS

forward-lookup function.

```
my @info = gethostbyname("example.com")
my @addrs = @info[4..$ #info]
@addrs |> map inet_ntoa |> e p
my $ip = inet_ntoa((gethostbyname("localhost"))[4])
p $ip   # 127.0.0.1
```

gethostbyaddr

Perform a reverse DNS lookup — given a packed binary IP address and address family, return the hostname associated with that address. Returns `($name, $aliases, $addrtype, $length, @addrs)` on success, or empty list if no PTR record exists.

```
my $packed = inet_aton("8.8.8.8")
my $name = (gethostbyaddr($packed, AF_INET))[0]
p $name   # dns.google
```

getprotobyname

Look up a network protocol by its name and return protocol information. Returns `($name, $aliases, $proto_number)`. The protocol number is what you pass to `socket` as the protocol argument. Common names include `tcp`, `udp`, and `icmp`.

```
my ($name, $aliases, $proto) = getprotobyname("tcp")
p "$name = protocol $proto"   # tcp = protocol 6
socket(my $raw, AF_INET, SOCK_RAW, (getprotobyname("icmp"))[2])
```

getservbyname

Look up a network service by its name and protocol, returning the port number and related information. Returns `($name, $aliases, $port, $proto)`. The port is in host byte order. This resolves well-known service names like `http`, `ssh`, or `smtp` to their port numbers portably.

```
my ($name, $aliases, $port) = getservbyname("http", "tcp")
p "$name => port $port"        # http => port 80
my $ssh_port = (getservbyname("ssh", "tcp"))[2]
p $ssh_port                    # 22
```

Crypto & Encoding

133 topics

sha256

sha256 (alias s256) computes the SHA-256 cryptographic hash of the input data and returns it as a 64-character lowercase hexadecimal string. SHA-256 is the most widely used hash function for data integrity verification, content addressing, and digital signatures. The Rust implementation is significantly faster than pure-Perl alternatives. Accepts strings or byte buffers.

```
p sha256 "hello world"   # b94d27b9934d3e08...
my $checksum = sha256 slurp "release.tar.gz"
p $checksum
rl("passwords.txt") |> map sha256 |> e p
```

sha224

sha224 (alias s224) computes the SHA-224 cryptographic hash and returns a 56-character hex string. SHA-224 is a truncated variant of SHA-256 that produces a shorter digest while maintaining strong collision resistance. It is sometimes preferred when storage or bandwidth for the hash value is constrained, such as in compact data structures or short identifiers.

```
p sha224 "hello world"   # 2f05477fc24bb4fa...
my $h = sha224(tj {key => "value"})
p $h
```

sha384

sha384 (alias s384) computes the SHA-384 cryptographic hash and returns a 96-character hex string. SHA-384 is a truncated variant of SHA-512 that offers a middle ground between SHA-256 and SHA-512 in both digest length and security margin. It is commonly used in TLS certificate fingerprints and government security standards that require larger-than-256-bit digests.

```
p sha384 "hello world"   # fdbd8e75a67f29f7...
my $sig = sha384(slurp "document.pdf")
spurt "document.pdf.sha384", $sig
```

sha512

sha512 (alias s512) computes the SHA-512 cryptographic hash and returns a 128-character hex string. SHA-512 provides the largest digest size in the SHA-2 family and is the strongest option when maximum collision resistance is needed. On 64-bit systems, SHA-512 is often faster than SHA-256 because it operates on 64-bit words natively. Use this for high-security applications or when you need a longer hash.

```
p sha512 "hello world"   # 309ecc489c12d6eb...
my $hash = sha512(slurp "firmware.bin");
p $hash;
my @hashes = map { sha512 } @files
```

sha1

sha1 (alias s1) computes the SHA-1 hash and returns a 40-character hex string. SHA-1 is considered cryptographically broken for collision resistance and should not be used for security-sensitive applications. However, it remains widely used for non-security purposes such as Git object IDs, cache keys, and deduplication checksums where collision attacks are not a concern.

```
p sha1 "hello world"   # 2aae6c35c94fcfb4...
my $git_id = sha1("blob " . length($content) . "\0" . $content)
p $git_id$1

Note: prefer SHA-256 for any security-related use case.
```

crc32

`crc32` computes the CRC-32 checksum of the input data and returns it as an unsigned 32-bit integer. CRC-32 is not a cryptographic hash — it is a fast error-detection code used in network protocols (Ethernet, ZIP, PNG), file integrity checks, and hash table bucketing. It is extremely fast compared to SHA functions, making it suitable for high-throughput deduplication or quick change detection where collision resistance is not required.

```
p crc32 "hello world"   # 222957957
my $chk = crc32(slurp "archive.zip");
p sprintf "0x%08x", $chk;  # hex representation
my @checksums = map { crc32 } @chunks
```

hmac_sha256

`hmac_sha256` (alias `hmac`) computes an HMAC-SHA256 message authentication code using the given data and secret key, returning a hex string. HMAC combines a cryptographic hash with a secret key to produce a signature that verifies both data integrity and authenticity. This is the standard mechanism for signing API requests (AWS, Stripe, GitHub webhooks), generating secure tokens, and verifying message authenticity.

```
my $sig = hmac_sha256 "request body", "my-secret-key"
p $sig
my $webhook_sig = hmac("POST /hook\n$body", $secret)
p $webhook_sig eq $expected ? "valid" : "tampered"
```

blake2b

`blake2b` (alias `b2b`) computes the BLAKE2b-512 cryptographic hash and returns a 128-character hex string. BLAKE2b is faster than SHA-256 while being at least as secure. It is used in Argon2 password hashing, libsodium, WireGuard, and many modern cryptographic protocols. Prefer BLAKE2b or BLAKE3 for new projects over SHA-256.

```
p blake2b "hello world"   # 021ced8799...
my $hash = b2b(slurp "large-file.bin");
my @hashes = map { blake2b } @messages
```

blake2s

`blake2s` (alias `b2s`) computes the BLAKE2s-256 cryptographic hash and returns a 64-character hex string. BLAKE2s is optimized for 8-32 bit platforms while BLAKE2b targets 64-bit. Use BLAKE2s when targeting embedded systems, WebAssembly, or when a 256-bit digest suffices.

```
p blake2s "hello world"   # 9aec6806...
my $checksum = b2s($firmware_blob)
```

blake3

`blake3` (alias `b3`) computes the BLAKE3 cryptographic hash and returns a 64-character hex string (256-bit). BLAKE3 is the latest evolution — it is parallelizable, even faster than BLAKE2, and suitable for hashing large files at maximum speed. It is the recommended hash function for new projects where legacy compatibility is not required.

```
p blake3 "hello world"   # d74981efa...
my $hash = b3(slurp "gigabyte-file.bin")  # fast!
```

argon2_hash

`argon2_hash` (alias `argon2`) hashes a password using Argon2id, the winner of the Password Hashing Competition. Returns a PHC-format string containing algorithm parameters, salt, and hash. Argon2 is memory-hard and resistant to GPU/ASIC attacks. Use this for user password storage — never use MD5/SHA for passwords.

```
my $hash = argon2_hash("user-password")
to_file("user.hash", $hash)
# later...
if (argon2_verify("user-password", $hash)) {
    p "login successful"
}
```

argon2_verify

`argon2_verify` verifies a password against an Argon2 PHC hash string. Returns 1 if the password matches, 0 otherwise. The verification automatically extracts parameters from the stored hash, so changing Argon2 settings for new hashes does not break existing ones.

```
my $stored = rl("user.hash")
if (argon2_verify($user_input, $stored)) {
    p "access granted"
} else {
    p "invalid password"
}
```

bcrypt_hash

`bcrypt_hash` (alias `bcrypt`) hashes a password using the bcrypt algorithm, returning a standard `$2b$...` format string. Bcrypt has been the industry standard for password hashing for over 20 years. While Argon2 is now preferred for new systems, bcrypt remains secure and widely deployed.

```
my $hash = bcrypt_hash("my-password")
p $hash  # $2b$12$...
if (bcrypt_verify("my-password", $hash)) {
    p "correct"
}
```

bcrypt_verify

`bcrypt_verify` verifies a password against a bcrypt hash string. Returns 1 if correct, 0 otherwise. The cost factor and salt are extracted from the stored hash automatically.

```
if (bcrypt_verify($password, $stored_hash)) {
    grant_access()
}
```

scrypt_hash

`scrypt_hash` (alias `scrypt`) hashes a password using the scrypt algorithm, returning a PHC-format string. Scrypt is memory-hard like Argon2 and is used in cryptocurrency (Litecoin) and some enterprise systems. It predates Argon2 but remains secure.

```
my $hash = scrypt_hash("password123")
if (scrypt_verify("password123", $hash)) {
    p "verified"
}
```

scrypt_verify

`scrypt_verify` verifies a password against a scrypt PHC hash. Returns 1 on match, 0 otherwise.

```
if (scrypt_verify($input, $stored)) {
    p "valid"
}
```

pbkdf2

`pbkdf2` (alias `pbkdf2_derive`) derives a cryptographic key from a password using PBKDF2-HMAC-SHA256. Returns a 64-character hex string (32 bytes). Takes password, salt, and optional iteration count (default 100,000). Use this when you need a fixed-length key from a password — for encryption keys, not password storage (use Argon2/bcrypt for that).

```
my $key = pbkdf2("password", "random-salt")
p $key  # 64 hex chars = 32 bytes
my $strong = pbkdf2("pass", $salt, 200_000)  # more iterations
```

random_bytes

`random_bytes` (alias `randbytes`) generates cryptographically secure random bytes using the OS CSPRNG. Returns a byte buffer of the specified length. Use for encryption keys, nonces, salts, and any security-sensitive randomness. Never use `rand()` for cryptographic purposes.

```
my $key = random_bytes(32)   # 256-bit key
my $nonce = random_bytes(12) # 96-bit nonce for AES-GCM
p hex_encode($key)
```

random_bytes_hex

`random_bytes_hex` (alias `randhex`) generates cryptographically secure random bytes and returns them as a hex string. Convenient when you need a random hex token or key without a separate hex_encode call.

```
my $token = random_bytes_hex(16)  # 32 hex chars
p $token   # 7a3f9b2c1d...
my $api_key = "sk_" . randhex(24)
```

aes_encrypt

`aes_encrypt` (alias `aes_enc`) encrypts plaintext using AES-256-GCM authenticated encryption. Takes a 32-byte key (use `random_bytes(32)` or `pbkdf2`). Returns a base64 string containing the nonce and ciphertext. AES-GCM provides both confidentiality and integrity — tampering is detected on decryption.

```
my $key = random_bytes(32)
my $cipher = aes_encrypt($key, "secret message")
p $cipher  # base64 encoded
my $plain = aes_decrypt($key, $cipher)
p $plain   # secret message
```

aes_decrypt

`aes_decrypt` (alias `aes_dec`) decrypts AES-256-GCM ciphertext produced by `aes_encrypt`. Takes the same 32-byte key and the base64 ciphertext. Dies if the key is wrong or the ciphertext was tampered with (authentication failure).

```
my $plain = aes_decrypt($key, $ciphertext)
p $plain
# wrong key or tampered data raises exception
```

chacha_encrypt

`chacha_encrypt` (alias `chacha_enc`) encrypts plaintext using ChaCha20-Poly1305 authenticated encryption. Takes a 32-byte key. Returns base64(nonce || ciphertext || tag). ChaCha20-Poly1305 is the modern alternative to AES-GCM — faster in software, constant-time, and used in TLS 1.3, WireGuard, and SSH.

```
my $key = random_bytes(32)
my $cipher = chacha_encrypt($key, "secret data")
my $plain = chacha_decrypt($key, $cipher)
p $plain
```

chacha_decrypt

chacha_decrypt (alias chacha_dec) decrypts ChaCha20-Poly1305 ciphertext. Takes the same 32-byte key and base64 ciphertext. Dies on wrong key or tampering.

```
my $plain = chacha_decrypt($key, $cipher)
p $plain
```

ed25519_keygen

ed25519_keygen (alias ed_keygen) generates an Ed25519 signing keypair. Returns [private_key_hex, public_key_hex]. Ed25519 is the modern standard for digital signatures — fast, secure, and used in SSH, GPG, and cryptocurrency. The private key is 32 bytes (64 hex), the public key is also 32 bytes.

```
my ($priv, $pub) = @{ ed25519_keygen() }
p "Private: $priv"
p "Public:  $pub"
my $sig = ed25519_sign($priv, "message")
```

ed25519_sign

ed25519_sign (alias ed_sign) signs a message with an Ed25519 private key. Takes the private key (hex) and message. Returns a 128-character hex signature (64 bytes). Signatures are deterministic — the same key+message always produces the same signature.

```
my $sig = ed25519_sign($private_key, "hello world")
p $sig   # 128 hex chars
```

ed25519_verify

ed25519_verify (alias ed_verify) verifies an Ed25519 signature. Takes public_key_hex, message, signature_hex. Returns 1 if valid, 0 if invalid. Never trust unsigned data in security contexts.

```
if (ed25519_verify($pub, "hello world", $sig)) {
    p "signature valid"
} else {
    p "FORGED!"
}
```

x25519_keygen

x25519_keygen (alias x_keygen) generates an X25519 key-exchange keypair. Returns [private_key_hex, public_key_hex]. X25519 is the modern Diffie-Hellman — used in TLS 1.3, Signal, WireGuard. Both parties generate keypairs and exchange public keys to derive a shared secret.

```
my ($my_priv, $my_pub) = @{ x25519_keygen() }
# send $my_pub to peer, receive $their_pub
my $shared = x25519_dh($my_priv, $their_pub)
```

x25519_dh

x25519_dh (alias x_dh) performs X25519 Diffie-Hellman key exchange. Takes your private key and their public key. Returns a 64-character hex shared secret. Both parties derive the same secret, which can be used as an encryption key.

```
my $shared = x25519_dh($my_private, $their_public)
my $key = hex_decode(substr($shared, 0, 64))  # use as AES key
```

base64_encode

base64_encode (alias b64e) encodes a string or byte buffer as a Base64 string using the standard alphabet (A-Z, a-z, 0-9, +, /). Base64 is the standard way to embed binary data in text-based formats like JSON, XML, email (MIME), and data URIs. The output length is always a multiple of 4, padded with = as needed. Use base64_decode to reverse the encoding.

```
my $encoded = base64_encode "hello world"
p $encoded   # aGVsbG8gd29ybGQ=
my $img_data = slurp "photo.png"
my $data_uri = "" . base64_encode($img_data)
p base64_decode(base64_encode("round trip"))  # round trip
```

base64_decode

`base64_decode` (alias `b64d`) decodes a Base64-encoded string back to its original bytes. It accepts standard Base64 with padding and is tolerant of line breaks within the input. This is essential for processing email attachments, decoding JWT payloads, extracting embedded images from data URIs, or reading any Base64-encoded field from an API response. Raises an exception on invalid Base64 input.

```
my $decoded = base64_decode "aGVsbG8gd29ybGQ="
p $decoded   # hello world
my $img = base64_decode($api_response->{avatar_b64})
spurt "avatar.png", $img
my $json = base64_decode($jwt_parts[1])
dd json_decode $json
```

hex_encode

`hex_encode` (alias `hxe`) converts a string or byte buffer into its lowercase hexadecimal representation, with two hex characters per input byte. This is useful for displaying binary data in a human-readable format, generating hex-encoded keys or IDs, logging raw bytes, or preparing data for protocols that use hex encoding. The output is always an even number of characters.

```
p hex_encode "hello"   # 68656c6c6f
my $raw = slurp "key.bin"
p hex_encode $raw
my $color = hex_encode chr(255) . chr(128) . chr(0)
p "#$color"   # #ff8000
```

hex_decode

`hex_decode` (alias `hxd`) converts a hexadecimal string back to its original bytes, interpreting every two hex characters as one byte. This is the inverse of `hex_encode` and is useful for parsing hex-encoded binary data from config files, network protocols, or cryptographic outputs. The input must have an even number of valid hex characters (0-9, a-f, A-F) or an exception is raised.

```
my $bytes = hex_decode "68656c6c6f"
p $bytes   # hello
my $key = hex_decode $env_hex_key
my $mac = hmac_sha256($data, hex_decode($secret_hex))
p hex_decode(hex_encode("round trip"))  # round trip
```

uuid

`uuid` generates a cryptographically random UUID version 4 string in the standard 8-4-4-4-12 hyphenated format. Each call produces a unique identifier suitable for database primary keys, correlation IDs, session tokens, temporary file names, or any situation requiring a globally unique identifier without coordination. The randomness comes from the OS CSPRNG.

```
my $id = uuid()
p $id   # e.g., 550e8400-e29b-41d4-a716-446655440000
my %record = (id => uuid(), name => "Alice", created => time)
my @ids = map { uuid() } 1:10
```

jwt_encode

`jwt_encode` creates a signed JSON Web Token from a payload hashref and a secret key. The default algorithm is HS256 (HMAC-SHA256), but you can specify an alternative as the third argument. JWTs are the standard for stateless authentication tokens, API authorization, and secure inter-service communication. The returned string contains the base64url-encoded header, payload, and signature separated by dots.

```
my $token = jwt_encode({sub => "user123", exp => time + 3600}, "my-secret")
p $token
my $admin = jwt_encode({role => "admin", iat => time}, $secret, "HS512")
# send as Authorization header
http_request(method => "GET", url => $api_url,
    headers => {Authorization => "Bearer $token"})
```

jwt_decode

`jwt_decode` verifies the signature of a JSON Web Token using the provided secret key and returns the decoded payload as a hashref. If the signature is invalid, the token has been tampered with, or it has expired (when an `exp` claim is present), the function raises an exception. This is the secure way to validate incoming JWTs from clients or other services — always use this over `jwt_decode_unsafe` in production.

```
my $payload = jwt_decode($token, "my-secret")
p $payload->{sub}   # user123
my $claims = jwt_decode($bearer_token, $secret)
if ($claims->{role} eq "admin") {
    p "admin access granted"
}
```

Note: raises an exception on expired tokens, invalid signatures, or malformed input.

jwt_decode_unsafe

`jwt_decode_unsafe` decodes a JSON Web Token and returns the payload as a hashref without verifying the signature. This is intentionally insecure and should only be used for debugging, logging, or inspecting token contents in development environments. Never use this to make authorization decisions in production — an attacker can stryke arbitrary payloads. The function still parses the JWT structure and base64-decodes the payload, but skips all cryptographic checks.

```
# debugging only — never use for auth
my $claims = jwt_decode_unsafe($token)
dd $claims
p $claims->{sub}   # inspect without needing the secret
my $exp = $claims->{exp}
p "Expires: " . datetime_from_epoch($exp)$1

Note: this function exists for debugging. Use `jwt_decode` with a secret for any security-relevant validation.
```

url_encode

Percent-encode a string so it is safe to embed in a URL query parameter or path segment. Unreserved characters (alphanumeric, `-`, `_`, `.`, `~`) are left as-is; everything else becomes `%XX`. The alias `uri_escape` matches the classic `URI::Escape` name for Perl muscle-memory.

```
my $q = "hello world & friends"
my $safe = url_encode($q)
p $safe   # hello%20world%20%26%20friends
my $url = "https://example.com/search?q=" . url_encode($q)
p $url$1

Note: does not encode the full URL structure — encode individual components, not the whole URL.
```

url_decode

Decode a percent-encoded string back to its original form, converting `%XX` sequences to the corresponding bytes and `+` to space. The alias `uri_unescape` matches `URI::Escape` conventions. Use this when parsing query strings from incoming URLs or reading URL-encoded form data.

```
my $encoded = "hello%20world%20%26%20friends"
p url_decode($encoded)   # hello world & friends
# round-trip
my $orig = "café □"
p url_decode(url_encode($orig)) eq $orig  # 1
```

gzip

Compress a string or byte buffer using the gzip (RFC 1952) format and return the compressed bytes. Useful for shrinking data before writing to disk or sending over the network. Pairs with `gunzip` for decompression. The compression level is chosen automatically for a good speed/size tradeoff.

```
my $raw = "hello world" x 1000
my $gz = gzip($raw)
to_file("data.gz", $gz)
p length($gz)       # much smaller than original
p gunzip($gz) eq $raw  # 1
```

gunzip

Decompress gzip-compressed data (RFC 1952) and return the original bytes. Dies if the input is not valid gzip. Use this to read `.gz` files or decompress data received from HTTP responses with `Content-Encoding: gzip`. Always the inverse of `gzip`.

```
my $compressed = rl("archive.gz")
my $text = gunzip($compressed)
p $text
# round-trip in a pipeline
"payload" |> gzip |> gunzip |> p  # payload
```

zstd

Compress a string or byte buffer using the Zstandard algorithm and return the compressed bytes. Zstandard offers significantly better compression ratios and speed compared to gzip, making it ideal for large datasets, IPC buffers, and caching. Pairs with `zstd_decode` for decompression.

```
my $big = "x]" x 100_000
my $compressed = zstd($big)
p length($compressed)  # fraction of original
to_file("data.zst", $compressed)
p zstd_decode($compressed) eq $big  # 1
```

zstd_decode

Decompress Zstandard-compressed data and return the original bytes. Dies if the input is not valid Zstandard. This is the inverse of `zstd`. Use it to read `.zst` files or decompress cached buffers that were compressed with `zstd`.

```
my $packed = zstd("important data\n" x 500)
my $original = zstd_decode($packed)
p $original
# file round-trip
to_file("cache.zst", zstd($payload))
p zstd_decode(rl("cache.zst"))
```

sha3_256

`sha3_256` (alias `s3_256`) computes the SHA3-256 hash (NIST FIPS 202) and returns a 64-character hex string. SHA-3 is the newest NIST-approved hash family, designed as a backup if SHA-2 is ever compromised. It uses the Keccak sponge construction which is fundamentally different from SHA-2.

```
p sha3_256 "hello world"   # 644bcc7e...
my $h = s3_256(slurp "file.bin")
```

sha3_512

`sha3_512` (alias `s3_512`) computes the SHA3-512 hash and returns a 128-character hex string. Provides a 512-bit digest with the SHA-3 sponge construction.

```
p sha3_512 "hello world"   # 840006...
```

shake128

`shake128` is a SHA-3 extendable-output function (XOF). Unlike fixed-length hashes, SHAKE can produce arbitrary-length output. Takes data and output length in bytes, returns hex. Useful for key derivation or when you need a variable-length digest.

```
p shake128("seed", 32)   # 64 hex chars (32 bytes)
p shake128("seed", 64)   # 128 hex chars
```

shake256

`shake256` is a SHA-3 XOF with higher security margin than SHAKE128. Takes data and output length in bytes, returns hex.

```
p shake256("seed", 32)
p shake256("key material", 64)
```

ripemd160

`ripemd160` (alias `rmd160`) computes the RIPEMD-160 hash and returns a 40-character hex string. RIPEMD-160 is used in Bitcoin addresses (Hash160 = RIPEMD160(SHA256(pubkey))) and some legacy systems. Not recommended for new designs but essential for Bitcoin/crypto compatibility.

```
p ripemd160 "hello world"   # 98c615784c...
my $hash160 = ripemd160(hex_decode(sha256($pubkey)))
```

siphash

`siphash` computes SipHash-2-4 with default keys (0,0) and returns a 16-character hex string (64-bit). SipHash is designed for hash table DoS resistance — it's fast and keyed, preventing attackers from crafting collisions. Used in Rust's HashMap, Python dict, and many other hash tables.

```
p siphash "key"   # 16 hex chars
```

hmac_sha1

`hmac_sha1` computes HMAC-SHA1 and returns a 40-character hex string. Used in OAuth 1.0, TOTP (Google Authenticator), and legacy APIs. Prefer HMAC-SHA256 for new designs.

```
p hmac_sha1("secret", "message")   # 40 hex chars
```

hmac_sha384

`hmac_sha384` computes HMAC-SHA384 and returns a 96-character hex string. Middle ground between SHA-256 and SHA-512.

```
p hmac_sha384("key", "data")   # 96 hex chars
```

hmac_sha512

`hmac_sha512` computes HMAC-SHA512 and returns a 128-character hex string. Maximum security margin in the SHA-2 family.

```
p hmac_sha512("key", "data")   # 128 hex chars
```

hmac_md5

`hmac_md5` computes HMAC-MD5 and returns a 32-character hex string. Legacy only — MD5 is broken for collision resistance but HMAC-MD5 is still considered safe for authentication. Only use for compatibility with old systems.

```
p hmac_md5("key", "data")   # 32 hex chars
```

hkdf_sha256

hkdf_sha256 (alias hkdf) is HKDF key derivation (RFC 5869) using HMAC-SHA256. Extracts entropy from input key material and expands it to the desired length. Used to derive encryption keys from shared secrets (after ECDH/X25519). Args: ikm, salt, info, output_length.

```
my $key = hkdf("shared_secret", "salt", "context", 32)  # 64 hex
my $enc_key = hex_decode($key)  # 32 bytes for AES-256
```

hkdf_sha512

hkdf_sha512 is HKDF using HMAC-SHA512. Higher security margin than SHA256 variant.

```
my $key = hkdf_sha512("ikm", "salt", "info", 64)  # 128 hex chars
```

poly1305

poly1305 computes a Poly1305 one-time MAC. Takes a 32-byte key and message, returns a 32-character hex tag (128-bit). Poly1305 is used with ChaCha20 in TLS 1.3. CRITICAL: each key must only be used once — reusing a key completely breaks security.

```
my $key = random_bytes(32)
p poly1305($key, "message")   # 32 hex chars
```

rsa_keygen

rsa_keygen generates an RSA keypair. Takes key size in bits (2048, 3072, or 4096). Returns [private_key_pem, public_key_pem]. RSA is the most widely used asymmetric algorithm — essential for TLS, SSH, and JWT RS256 signing.

```
my @kp = rsa_keygen(2048)
my ($priv, $pub) = @kp
spurt("private.pem", $priv)
spurt("public.pem", $pub)
```

rsa_encrypt

rsa_encrypt (alias rsa_enc) encrypts data with RSA-OAEP-SHA256. Takes public_key_pem and plaintext. Returns base64 ciphertext. Message size is limited to key_size/8 - 66 bytes (e.g. 190 bytes for 2048-bit key).

```
my $cipher = rsa_encrypt($pub_pem, "secret")
my $plain = rsa_decrypt($priv_pem, $cipher)
```

rsa_decrypt

rsa_decrypt (alias rsa_dec) decrypts RSA-OAEP ciphertext. Takes private_key_pem and base64 ciphertext.

```
my $plain = rsa_decrypt($priv, $ciphertext)
p $plain
```

rsa_sign

rsa_sign signs a message with RSA-PKCS1v15-SHA256. Takes private_key_pem and message. Returns base64 signature. This is the RS256 algorithm used in JWT.

```
my $sig = rsa_sign($priv, "message")
if (rsa_verify($pub, "message", $sig)) {
    p "valid"
}
```

rsa_verify

rsa_verify verifies an RSA-PKCS1v15-SHA256 signature. Takes public_key_pem, message, and base64 signature. Returns 1 if valid, 0 if not.

```
if (rsa_verify($pub_pem, $msg, $sig)) {
    p "signature valid"
}
```

ecdsa_p256_keygen

ecdsa_p256_keygen (alias p256_keygen) generates an ECDSA P-256 (secp256r1/prime256v1) keypair. Returns [private_hex, public_hex_compressed]. P-256 is the NIST curve used in TLS, ES256 JWT, and WebAuthn.

```
my @kp = ecdsa_p256_keygen()
my ($priv, $pub) = @kp
```

ecdsa_p256_sign

ecdsa_p256_sign (alias p256_sign) signs a message with ECDSA P-256. Takes private_key_hex and message. Returns DER-encoded signature as hex.

```
my $sig = ecdsa_p256_sign($priv, "hello")
```

ecdsa_p256_verify

ecdsa_p256_verify (alias p256_verify) verifies an ECDSA P-256 signature. Returns 1 if valid.

```
if (ecdsa_p256_verify($pub, "hello", $sig)) {
    p "valid"
}
```

ecdsa_p384_keygen

ecdsa_p384_keygen generates an ECDSA P-384 keypair. P-384 offers more security margin than P-256.

```
my @kp = ecdsa_p384_keygen()
```

ecdsa_p384_sign

ecdsa_p384_sign signs with ECDSA P-384.

```
my $sig = ecdsa_p384_sign($priv, $msg)
```

ecdsa_p384_verify

ecdsa_p384_verify verifies an ECDSA P-384 signature.

```
p ecdsa_p384_verify($pub, $msg, $sig)
```

ecdsa_secp256k1_keygen

ecdsa_secp256k1_keygen generates an ECDSA secp256k1 keypair. This is the Bitcoin/Ethereum curve — different from P-256.

```
my @kp = ecdsa_secp256k1_keygen()
```

ecdsa_secp256k1_sign

ecdsa_secp256k1_sign signs with ECDSA secp256k1.

```
my $sig = ecdsa_secp256k1_sign($priv, $msg)
```

ecdsa_secp256k1_verify

ecdsa_secp256k1_verify verifies an ECDSA secp256k1 signature.

```
p ecdsa_secp256k1_verify($pub, $msg, $sig)
```

ecdh_p256

ecdh_p256 (alias p256_dh) performs ECDH key exchange on P-256. Takes my_private_hex and their_public_hex, returns shared_secret_hex. Use HKDF to derive encryption keys from the shared secret.

```
my @alice = ecdsa_p256_keygen()
my @bob = ecdsa_p256_keygen()
my $shared_a = ecdh_p256($alice[0], $bob[1])
my $shared_b = ecdh_p256($bob[0], $alice[1])
p $shared_a eq $shared_b  # 1 — same secret
```

ecdh_p384

ecdh_p384 performs ECDH key exchange on P-384.

```
my $shared = ecdh_p384($my_priv, $their_pub)
```

base32_encode

base32_encode (alias b32e) encodes data as RFC 4648 Base32. Used in TOTP secrets, onion addresses, and Bech32. Returns uppercase with padding.

```
p base32_encode("hello")   # NB5WY3DP
my $secret = base32_encode(random_bytes(20))
```

base32_decode

base32_decode (alias b32d) decodes RFC 4648 Base32 back to bytes. Accepts with or without padding.

```
p base32_decode("NB5WY3DP")   # hello
```

base58_encode

base58_encode (alias b58e) encodes data using Bitcoin's Base58 alphabet (no 0, O, I, l to avoid confusion). Used in Bitcoin addresses, IPFS CIDs.

```
p base58_encode("hello")   # Cn8eVZg
```

base58_decode

base58_decode (alias b58d) decodes Base58 back to bytes.

```
p base58_decode("Cn8eVZg")   # hello
```

totp

totp (alias totp_generate) generates a TOTP code (RFC 6238) for 2FA. Takes base32-encoded secret, optional digits (default 6), optional period (default 30s). Compatible with Google Authenticator, Authy, etc.

```
my $secret = base32_encode(random_bytes(20))
my $code = totp($secret)
p $code   # 6-digit code
# Custom: 8 digits, 60s period
my $code8 = totp($secret, 8, 60)
```

totp_verify

`totp_verify` verifies a TOTP code with optional time window (default ±1 period). Returns 1 if valid.

```
if (totp_verify($secret, $user_code)) {
    p "2FA valid"
}
# Wider window: ±2 periods
totp_verify($secret, $code, 2)
```

hotp

`hotp` (alias `hotp_generate`) generates an HOTP code (RFC 4226) using a counter. Takes base32 secret, counter value, optional digits.

```
my $code = hotp($secret, 42)  # counter=42
```

aes_cbc_encrypt

`aes_cbc_encrypt` (alias `aes_cbc_enc`) encrypts with AES-256-CBC and PKCS7 padding. Takes 32-byte key, plaintext, optional 16-byte IV (auto-generated if omitted). Returns base64(iv || ciphertext). Legacy mode — prefer `aes_encrypt` (GCM) for new code.

```
my $key = random_bytes(32)
my $ct = aes_cbc_encrypt($key, "secret")
my $pt = aes_cbc_decrypt($key, $ct)
```

aes_cbc_decrypt

`aes_cbc_decrypt` (alias `aes_cbc_dec`) decrypts AES-256-CBC. Takes 32-byte key and base64(iv || ciphertext).

```
my $pt = aes_cbc_decrypt($key, $ciphertext)
```

qr_ascii

`qr_ascii` (alias `qr`) generates a QR code as ASCII art. Perfect for terminal output.

```
p qr("https://example.com")
p qr("otpauth://totp/App:user?secret=$secret&issuer=App")
```

qr_png

`qr_png` generates a QR code as PNG image data (base64 encoded). Optional size parameter. Save to file or embed in HTML.

```
my $png = qr_png("https://example.com")
spurt("qr.png", base64_decode($png))
# Larger QR
my $big = qr_png($url, 16)
```

qr_svg

`qr_svg` generates a QR code as SVG string. Scalable vector graphics, ideal for web.

```
my $svg = qr_svg("https://example.com")
spurt("qr.svg", $svg)
```

barcode_code128

barcode_code128 (alias code128) generates a Code 128 barcode as ASCII. Code 128 supports alphanumeric data and is widely used in shipping labels.

```
p code128("ABC-123")
```

barcode_code39

barcode_code39 (alias code39) generates a Code 39 barcode. Supports uppercase, digits, and some symbols. Used in automotive and defense.

```
p code39("HELLO123")
```

barcode_ean13

barcode_ean13 (alias ean13) generates an EAN-13 barcode (European Article Number). Standard retail barcode — requires exactly 12-13 digits.

```
p ean13("5901234123457")
```

barcode_svg

barcode_svg generates a barcode as SVG. Second argument specifies type: code128, code39, ean13, upca.

```
my $svg = barcode_svg("ABC-123", "code128")
spurt("barcode.svg", $svg)
my $retail = barcode_svg("012345678905", "upca")
```

brotli

brotli (alias br) compresses data using the Brotli algorithm (RFC 7932). Excellent compression ratio, used in HTTP compression. Returns compressed bytes.

```
my $compressed = brotli($data)
p length($data) . " -> " . length($compressed)
```

brotli_decode

brotli_decode (alias ubr) decompresses Brotli data.

```
my $original = brotli_decode($compressed)
```

xz

xz (alias lzma) compresses data using XZ/LZMA2. Best compression ratio, slower. Returns compressed bytes.

```
my $compressed = xz($data)
spurt("file.xz", $compressed)
```

xz_decode

xz_decode (aliases unxz, unlzma) decompresses XZ/LZMA data.

```
my $original = xz_decode(slurp("file.xz"))
```

bzip2

bzip2 (alias bz2) compresses data using bzip2. Good compression, moderate speed. Returns compressed bytes.

```
my $compressed = bzip2($data)
```

bzip2_decode

`bzip2_decode` (aliases `bunzip2`, `ubz2`) decompresses bzip2 data.

```
my $original = bunzip2($compressed)
```

lz4

`lz4` compresses data using LZ4. Very fast compression/decompression, moderate ratio. Ideal for real-time compression.

```
my $compressed = lz4($data)
```

lz4_decode

`lz4_decode` (alias `unlz4`) decompresses LZ4 data.

```
my $original = lz4_decode($compressed)
```

snappy

`snappy` (alias `snp`) compresses data using Snappy. Fastest compression, used in databases and RPC. Returns compressed bytes.

```
my $compressed = snappy($data)
```

snappy_decode

`snappy_decode` (alias `unsnappy`) decompresses Snappy data.

```
my $original = snappy_decode($compressed)
```

lzw

`lzw` compresses data using LZW (GIF/TIFF style). Classic algorithm.

```
my $compressed = lzw($data)
```

lzw_decode

`lzw_decode` (alias `unlzw`) decompresses LZW data.

```
my $original = lzw_decode($compressed)
```

tar_create

`tar_create` (alias `tar`) creates a tar archive from a directory. Returns tar bytes.

```
my $archive = tar_create("./src")
spurt("backup.tar", $archive)
```

tar_extract

`tar_extract` (alias `untar`) extracts a tar archive to a directory.

```
tar_extract(slurp("backup.tar"), "./restored")
```

tar_list

`tar_list` lists files in a tar archive. Returns array of paths.

```
my @files = @{tar_list(slurp("backup.tar"))}
@files |> e p
```

tar_gz_create

`tar_gz_create` (alias `tgz`) creates a gzipped tar archive. Convenience for tar + gzip.

```
my $tgz = tgz("./project")
spurt("project.tar.gz", $tgz)
```

tar_gz_extract

`tar_gz_extract` (alias `untgz`) extracts a .tar.gz archive.

```
untgz(slurp("project.tar.gz"), "./extracted")
```

zip_create

`zip_create` creates a ZIP archive from a directory. Returns zip bytes.

```
my $archive = zip_create("./docs")
spurt("docs.zip", $archive)
```

zip_extract

`zip_extract` extracts a ZIP archive to a directory.

```
zip_extract(slurp("docs.zip"), "./extracted")
```

zip_list

`zip_list` lists files in a ZIP archive. Returns array of paths.

```
my @files = @{zip_list(slurp("archive.zip"))}
```

md4

`md4` computes the MD4 hash and returns a 32-character hex string. MD4 is completely broken — only use for legacy NTLM compatibility or historical systems.

```
p md4("hello")   # 32 hex chars
```

xxh32

`xxh32` (alias `xxhash32`) computes xxHash32 — extremely fast non-cryptographic hash. Optional seed parameter. Returns 8 hex chars.

```
p xxh32("hello")        # default seed 0
p xxh32("hello", 42)    # with seed
```

xxh64

`xxh64` (alias `xxhash64`) computes xxHash64 — fast 64-bit hash. Returns 16 hex chars.

```
p xxh64("hello")
```

xxh3

`xxh3` (alias `xxhash3`) computes xxHash3-64 — newest xxHash variant, fastest on modern CPUs. Returns 16 hex chars.

```
p xxh3("hello")
```

xxh3_128

`xxh3_128` computes xxHash3-128. Returns 32 hex chars.

```
p xxh3_128("hello")
```

murmur3

`murmur3` (alias `murmur3_32`) computes MurmurHash3 32-bit. Fast non-cryptographic hash, widely used in hash tables and bloom filters. Optional seed. Returns 8 hex chars.

```
p murmur3("hello")       # default seed 0
p murmur3("hello", 42)   # with seed
```

murmur3_128

`murmur3_128` computes MurmurHash3 128-bit (x64 variant). Returns 32 hex chars.

```
p murmur3_128("hello")
```

blowfish_encrypt

`blowfish_encrypt` (alias `bf_enc`) encrypts with Blowfish-CBC. Key=4-56 bytes, optional 8-byte IV. Legacy cipher — use AES for new code.

```
my $ct = blowfish_encrypt($key, "secret")
my $pt = blowfish_decrypt($key, $ct)
```

blowfish_decrypt

`blowfish_decrypt` (alias `bf_dec`) decrypts Blowfish-CBC.

```
my $pt = blowfish_decrypt($key, $ciphertext)
```

des3_encrypt

`des3_encrypt` (aliases `3des_enc`, `tdes_enc`) encrypts with Triple DES (3DES) CBC. Key=24 bytes (three 8-byte DES keys). Legacy cipher for PCI-DSS compliance.

```
my $key = random_bytes(24)
my $ct = des3_encrypt($key, "secret")
my $pt = des3_decrypt($key, $ct)
```

des3_decrypt

`des3_decrypt` (aliases `3des_dec`, `tdes_dec`) decrypts Triple DES (3DES) CBC.

```
my $pt = des3_decrypt($key, $ciphertext)
```

twofish_encrypt

`twofish_encrypt` (alias `tf_enc`) encrypts with Twofish-CBC. Key=16/24/32 bytes. AES finalist, still secure.

```
my $key = random_bytes(32)
my $ct = twofish_encrypt($key, "secret")
```

twofish_decrypt

twofish_decrypt (alias tf_dec) decrypts Twofish-CBC.

```
my $pt = twofish_decrypt($key, $ct)
```

camellia_encrypt

camellia_encrypt (alias cam_enc) encrypts with Camellia-CBC. Key=16/24/32 bytes. Japanese/EU standard, equivalent security to AES.

```
my $ct = camellia_encrypt($key, "secret")
```

camellia_decrypt

camellia_decrypt (alias cam_dec) decrypts Camellia-CBC.

```
my $pt = camellia_decrypt($key, $ct)
```

cast5_encrypt

cast5_encrypt encrypts with CAST5-CBC. Key=5-16 bytes. Used in PGP.

```
my $ct = cast5_encrypt($key, "secret")
```

cast5_decrypt

cast5_decrypt decrypts CAST5-CBC.

```
my $pt = cast5_decrypt($key, $ct)
```

salsa20

salsa20 encrypts with Salsa20 stream cipher. Key=32 bytes, nonce auto-generated. Fast, secure stream cipher.

```
my $ct = salsa20($key, "data")
my $pt = salsa20_decrypt($key, $ct)
```

salsa20_decrypt

salsa20_decrypt decrypts Salsa20.

```
my $pt = salsa20_decrypt($key, $ct)
```

xsalsa20

xsalsa20 encrypts with XSalsa20 (extended 24-byte nonce). Safer for random nonces.

```
my $ct = xsalsa20($key, "data")
```

xsalsa20_decrypt

xsalsa20_decrypt decrypts XSalsa20.

```
my $pt = xsalsa20_decrypt($key, $ct)
```

secretbox

`secretbox` (alias `secretbox_seal`) is NaCl's symmetric authenticated encryption (XSalsa20-Poly1305). Key=32 bytes. Simple, secure, fast.

```
my $key = random_bytes(32)
my $ct = secretbox($key, "message")
my $pt = secretbox_open($key, $ct)
```

secretbox_open

`secretbox_open` decrypts and authenticates NaCl secretbox.

```
my $pt = secretbox_open($key, $ct)
```

nacl_box_keygen

`nacl_box_keygen` generates a NaCl box keypair (X25519). Returns [secret_key_hex, public_key_hex].

```
my @kp = nacl_box_keygen()
my ($sk, $pk) = @kp
```

nacl_box

`nacl_box` is NaCl's asymmetric authenticated encryption. Takes recipient's public key, sender's secret key, plaintext.

```
my @alice = nacl_box_keygen()
my @bob = nacl_box_keygen()
my $ct = nacl_box($bob[1], $alice[0], "hello")  # to Bob from Alice
my $pt = nacl_box_open($alice[1], $bob[0], $ct)  # Bob decrypts
```

nacl_box_open

`nacl_box_open` decrypts NaCl box. Takes sender's public key, recipient's secret key, ciphertext.

```
my $pt = nacl_box_open($sender_pk, $my_sk, $ct)
```

Special Math Functions

12 topics

erf

`erf` computes the error function, which arises in probability, statistics, and solutions to the heat equation. erf(x) is the probability that a standard normal random variable falls in [-x√2, x√2]. Returns a value in (-1, 1).

```
p erf(0)      # 0
p erf(1)      # 0.8427...
p erf(10)     # ~1
```

erfc

`erfc` computes the complementary error function erfc(x) = 1 - erf(x). Numerically stable for large x where erf(x) ≈ 1. Used in computing tail probabilities of normal distributions.

```
p erfc(0)     # 1
p erfc(3)     # 0.0000220...
```

gamma

`gamma` (alias `tgamma`) computes the gamma function Γ(x), the extension of factorial to real numbers. Γ(n) = (n-1)! for positive integers. Used throughout statistics, physics, and combinatorics.

```
p gamma(5)    # 24 (= 4!)
p gamma(0.5)  # √π ≈ 1.7724...
p gamma(1)    # 1
```

lgamma

`lgamma` (alias `ln_gamma`) computes the natural logarithm of the gamma function: ln(Γ(x)). Avoids overflow for large arguments where Γ(x) would be astronomical. Essential for computing log-probabilities in statistics.

```
p lgamma(100)  # 359.13...
p lgamma(1000) # 5905.22...
```

digamma

`digamma` (alias `psi`) computes the digamma function ψ(x) = d/dx ln(Γ(x)), the logarithmic derivative of gamma. Appears in Bayesian statistics (expected log of Dirichlet variables), optimization, and special function theory.

```
p digamma(1)   # -□ ≈ -0.5772 (Euler-Mascheroni)
p digamma(2)   # 1 - □ ≈ 0.4228
```

beta_fn

`beta_fn` computes the beta function B(a,b) = Γ(a)Γ(b)/Γ(a+b). The beta function is the normalizing constant of the Beta distribution and appears throughout Bayesian statistics and combinatorics.

```
p beta_fn(2, 3)   # 0.0833...
p beta_fn(0.5, 0.5)  # π
```

lbeta

`lbeta` (alias `ln_beta`) computes ln(B(a,b)), the log of the beta function. Avoids overflow for small a or b where B(a,b) is very large.

```
p lbeta(0.01, 0.01)  # large positive
```

betainc

`betainc` (alias `beta_reg`) computes the regularized incomplete beta function I_x(a,b), the CDF of the Beta distribution. Takes (x, a, b). Essential for computing p-values of F-tests, t-tests, and binomial probabilities.

```
p betainc(0.5, 2, 3)  # P(Beta(2,3) < 0.5)
```

gammainc

`gammainc` (alias `gamma_li`) computes the lower incomplete gamma function γ(a,x) = ∫₀ˣ t^(a-1) e^(-t) dt. Used in computing CDFs of gamma and chi-squared distributions.

```
p gammainc(2, 1)  # Γ(2, 1)
```

gammaincc

`gammaincc` (alias `gamma_ui`) computes the upper incomplete gamma function Γ(a,x) = ∫ₓ^∞ t^(a-1) e^(-t) dt = Γ(a) - γ(a,x). Useful for tail probabilities.

```
p gammaincc(2, 1)  # Γ(2, 1)
```

gammainc_reg

`gammainc_reg` (alias `gamma_lr`) computes the regularized lower incomplete gamma P(a,x) = γ(a,x)/Γ(a), the CDF of the gamma distribution Gamma(a, 1).

```
p gammainc_reg(2, 1)  # P(Gamma(2,1) < 1)
```

gammaincc_reg

`gammaincc_reg` (alias `gamma_ur`) computes the regularized upper incomplete gamma Q(a,x) = 1 - P(a,x), the survival function of the gamma distribution.

```
p gammaincc_reg(2, 1)  # P(Gamma(2,1) > 1)
```

Bessel / Airy / Hankel / Struve / Kelvin

19 topics

bessel_j0

`bessel_j0` (alias `j0`) computes the Bessel function of the first kind, order 0.

```
p j0(0)    # 1.0
p j0(2.4)  # ≈ 0 (first zero)
```

bessel_j1

`bessel_j1` (alias `j1`) computes the Bessel function of the first kind, order 1.

```
p j1(0)    # 0.0
p j1(1.84) # ≈ 0.582 (first max)
```

bessel_j

`bessel_j N, X` — Bessel function of the first kind J_n(x) for integer n. Generalises `bessel_j0` / `bessel_j1`.

```
p bessel_j(2, 5)   # ≈ 0.0466
p bessel_j(0, 0)   # 1
```

bessel_y

`bessel_y N, X` — Bessel function of the second kind Y_n(x). Singular at x = 0.

```
p bessel_y(0, 1)   # ≈ 0.0883
p bessel_y(1, 5)   # ≈ 0.1479
```

bessel_i

`bessel_i N, X` — Modified Bessel function I_n(x). Grows exponentially for large x.

```
p bessel_i(0, 1)   # ≈ 1.266
p bessel_i(2, 3)   # ≈ 2.245
```

bessel_k

`bessel_k N, X` — Modified Bessel function K_n(x). Decays exponentially for large x; singular at 0.

```
p bessel_k(0, 1)   # ≈ 0.4210
p bessel_k(1, 1)   # ≈ 0.6019
```

hankel_h1

`hankel_h1 N, X` returns the complex Hankel H_n^(1)(x) = J_n(x) + i Y_n(x) as `[Re, Im]`.

```
my ($re, $im) = hankel_h1(0, 1)
p "$re + i*$im"
```

hankel_h2

`hankel_h2 N, X` returns H_n^(2)(x) = J_n(x) - i Y_n(x) as `[Re, Im]`.

bessel_j_zero

`bessel_j_zero N, K` computes the Kth positive zero of J_n. Uses McMahon expansion + Newton refinement.

```
p bessel_j_zero(0, 1)  # 2.4048…
p bessel_j_zero(1, 1)  # 3.8317…
```

airy_ai

`airy_ai X` — Airy function Ai(x) (decaying solution of y'' = x y).

```
p airy_ai(0)   # ≈ 0.3550
p airy_ai(2)   # ≈ 0.0349
```

airy_bi

`airy_bi X` — Airy function Bi(x) (growing solution).

airy_ai_prime

`airy_ai_prime X` — derivative Ai'(x).

airy_bi_prime

`airy_bi_prime X` — derivative Bi'(x).

spherical_bessel_j

`spherical_bessel_j N, X` — j_n(x) = √(π/2x) J_{n+1/2}(x).

```
p spherical_bessel_j(0, 1)  # sin(1)
```

spherical_bessel_y

`spherical_bessel_y N, X` — y_n(x) = √(π/2x) Y_{n+1/2}(x).

struve_h

`struve_h N, X` — Struve function H_n(x) (solution of an inhomogeneous Bessel equation).

struve_l

`struve_l N, X` — Modified Struve L_n(x).

kelvin_ber

`kelvin_ber X` — Real part of J_0(x e^(3πi/4)) (Kelvin function ber_0).

kelvin_bei

`kelvin_bei X` — Imaginary part of J_0(x e^(3πi/4)) (Kelvin function bei_0).

Orthogonal Polynomials

13 topics

legendre_p

`legendre_p N, X` — Legendre polynomial P_n(x). Uses the stable Bonnet recurrence (2n+1) x P_n = (n+1) P_{n+1} + n P_{n-1}.

```
p legendre_p(2, 0.5)  # -0.125
p legendre_p(5, 1)    # 1
```

legendre_q

`legendre_q N, X` — Legendre Q_n(x), the second-kind solution. NaN for |x| ? 1.

assoc_legendre_p

`assoc_legendre_p N, M, X` — Associated Legendre P_n^m(x). Building block for spherical harmonics.

hermite_h

`hermite_h N, X` — Physicist's Hermite H_n(x): H_0=1, H_1=2x, H_{n+1}=2x H_n - 2n H_{n-1}.

```
p hermite_h(3, 1)   # -4
```

hermite_he

`hermite_he N, X` — Probabilist's Hermite He_n(x): He_0=1, He_1=x, He_{n+1}=x He_n - n He_{n-1}.

laguerre_l

`laguerre_l N, X` — Laguerre polynomial L_n(x).

assoc_laguerre_l

`assoc_laguerre_l N, ALPHA, X` — Generalised Laguerre L_n^α(x).

jacobi_p

`jacobi_p N, ALPHA, BETA, X` — Jacobi polynomial P_n^{(α,β)}(x). Generalises Legendre/Chebyshev/Gegenbauer.

gegenbauer_c

`gegenbauer_c N, ALPHA, X` — Gegenbauer C_n^α(x). Reduces to Chebyshev/Legendre at specific α.

chebyshev_t

`chebyshev_t N, X` — Chebyshev polynomial of the first kind T_n(x). Recurrence T_{n+1} = 2x T_n - T_{n-1}.

```
p chebyshev_t(3, 0.5)   # -0.5
```

chebyshev_u

`chebyshev_u N, X` — Chebyshev polynomial of the second kind U_n(x).

spherical_harmonic_y

`spherical_harmonic_y L, M, THETA, PHI` — Spherical harmonic Y_l^m(θ, φ) returned as `[Re, Im]`.

zernike_r

`zernike_r N, M, R` — Radial Zernike polynomial R_n^m(r). 0 if (n − m) is odd or m > n.

Elliptic Integrals & Functions

18 topics

elliptic_k

`elliptic_k M` — Complete elliptic integral K(m), m = k². Computed via Carlson R_F.

```
p elliptic_k(0)    # π/2
p elliptic_k(0.5)  # ≈ 1.8541
```

elliptic_e

`elliptic_e M` — Complete elliptic integral of the second kind E(m).

elliptic_pi

`elliptic_pi N, M` — Complete elliptic integral of the third kind Π(n | m).

elliptic_f

`elliptic_f PHI, M` — Incomplete F(φ, m).

elliptic_e_inc

`elliptic_e_inc PHI, M` — Incomplete E(φ, m).

elliptic_pi_inc

`elliptic_pi_inc N, PHI, M` — Incomplete Π(n; φ | m).

carlson_rf

`carlson_rf X, Y, Z` — Carlson symmetric form R_F. Building block for the elliptic family.

carlson_rd

`carlson_rd X, Y, Z` — Carlson R_D.

carlson_rj

`carlson_rj X, Y, Z, P` — Carlson R_J.

jacobi_sn

`jacobi_sn U, M` — Jacobi elliptic sn(u, m). Reduces to sin(u) at m = 0 and tanh(u) at m = 1.

```
p jacobi_sn(1, 0.5)  # ≈ 0.8030
```

jacobi_cn

`jacobi_cn U, M` — Jacobi elliptic cn(u, m). Reduces to cos(u) at m = 0.

jacobi_dn

`jacobi_dn U, M` — Jacobi elliptic dn(u, m). Reduces to 1 at m = 0.

jacobi_am

`jacobi_am U, M` — Jacobi amplitude φ such that sn(u, m) = sin(φ).

elliptic_theta

`elliptic_theta J, Z, Q` — Jacobi theta θ_j(z, q) for j ∈ {1, 2, 3, 4}, |q| < 1.

weierstrass_p

`weierstrass_p Z, G2, G3` — Weierstrass ℘(z; g_2, g_3) via Laurent series. Pole at z = 0.

weierstrass_zeta

`weierstrass_zeta Z, G2, G3` — Weierstrass ζ(z; g_2, g_3).

weierstrass_sigma

`weierstrass_sigma Z, G2, G3` — Weierstrass σ(z; g_2, g_3).

inverse_jacobi_sn

`inverse_jacobi_sn X, M` — inverse of `jacobi_sn`: returns u with sn(u, m) = x.

Zeta / Polylog / Lerch

9 topics

zeta

`zeta S` — Riemann ζ(s) via Euler-Maclaurin (s > 0.5) or reflection identity.

```
p zeta(2)    # π²/6 ≈ 1.6449
p zeta(4)    # ≈ 1.0823
p zeta(-1)   # -1/12
```

hurwitz_zeta

`hurwitz_zeta S, A` — Hurwitz ζ(s, a). `hurwitz_zeta(s, 1)` ≡ `zeta(s)`.

polylog

`polylog N, Z` — Polylogarithm Li_n(z) = Σ z^k / k^n for |z| < 1.

```
p polylog(2, 1)    # π²/6
```

dilog

`dilog Z` — Dilogarithm Li_2(z). Convenience alias for `polylog(2, z)`.

lerch_phi

`lerch_phi Z, S, A` — Lerch transcendent Φ(z, s, a) = Σ z^k / (a+k)^s.

riemann_siegel_z

`riemann_siegel_z T` — Riemann-Siegel Z(t) on the critical line.

riemann_siegel_theta

`riemann_siegel_theta T` — Riemann-Siegel θ(t) (asymptotic Stirling form).

dirichlet_eta

`dirichlet_eta S` — Dirichlet η(s) = (1 − 2^(1−s)) ζ(s).

dirichlet_beta

`dirichlet_beta S` — Dirichlet β(s) = Σ (-1)^k / (2k+1)^s.

```
p dirichlet_beta(1)   # π/4 (Catalan-like)
p dirichlet_beta(2)   # Catalan's constant ≈ 0.9160
```

Hypergeometric

5 topics

hypergeometric_2f1

`hypergeometric_2f1 A, B, C, Z` — ${}_2F_1(a, b; c; z)$ Gauss hypergeometric. Series convergent for $|z| < 1$.

```
p hypergeometric_2f1(0.5, 0.5, 1, 0.5)  # ≈ 1.1803
```

hypergeometric_1f1

`hypergeometric_1f1 A, B, Z` — ${}_1F_1(a; b; z)$ Kummer / confluent hypergeometric.

hypergeometric_0f1

`hypergeometric_0f1 B, Z` — ${}_0F_1(; b; z)$. Related to Bessel: $J_\nu(x) = (x/2)^\nu / \Gamma(\nu+1) \cdot {}_0F_1(; \nu+1; -x^2/4)$.

hypergeometric_pfq

`hypergeometric_pfq AS, BS, Z` — generalised _pF_q. AS, BS are arrays of upper/lower parameters.

hypergeometric_u

`hypergeometric_u A, B, Z` — Tricomi's confluent U(a, b, z).

Modular Forms

4 topics

dedekind_eta

`dedekind_eta Y` — Dedekind η(iy) on the imaginary axis. Real-valued q-series with q = exp(-2π y).

klein_j

`klein_j Y` — Klein j-invariant j(iy) = E_4^3 / Δ.

modular_lambda

`modular_lambda Y` — Modular λ(iy) = θ_2(0, q)^4 / θ_3(0, q)^4.

ramanujan_tau

`ramanujan_tau N` — τ(n), the nth coefficient of Δ(τ) = η(τ)^24. Returns an integer.

```
p ramanujan_tau(1)   # 1
p ramanujan_tau(2)   # -24
p ramanujan_tau(3)   # 252
```

Si / Ci / Ei / Li / Fresnel Integrals

9 topics

sin_integral

`sin_integral X` — Si(x) = ∫₀ˣ sin(t)/t dt. Si(∞) = π/2.

```
p sin_integral(1)    # ≈ 0.9461
p sin_integral(10)   # ≈ 1.6583
```

cos_integral

`cos_integral X` — Ci(x) = γ + ln(x) + ∫₀ˣ (cos(t)−1)/t dt. NaN for x ≤ 0.

sinh_integral

`sinh_integral X` — Shi(x) = ∫₀ˣ sinh(t)/t dt.

cosh_integral

`cosh_integral X` — Chi(x) = γ + ln(x) + ∫₀ˣ (cosh(t)−1)/t dt.

exp_integral_e

`exp_integral_e N, X` — generalised exponential integral E_n(x) = ∫_1^∞ e^{-x t} / t^n dt.

exp_integral_ei

`exp_integral_ei X` — Ei(x). For x > 0 uses series + asymptotic split.

log_integral

`log_integral X` — li(x) = Ei(ln x). Approximates the prime-counting function near `prime_pi`.

```
p log_integral(1000)   # ≈ 178
```

fresnel_s

`fresnel_s X` — Fresnel S(x) = ∫₀ˣ sin(π t² / 2) dt.

fresnel_c

`fresnel_c X` — Fresnel C(x) = ∫₀ˣ cos(π t² / 2) dt.

Number Theory (Wolfram Parity)

13 topics

jacobi_symbol

`jacobi_symbol A, N` — Jacobi symbol (a/n) for positive odd n. Returns -1, 0, or 1.

```
p jacobi_symbol(2, 7)  # 1
```

kronecker_symbol

`kronecker_symbol A, N` — Extension of the Jacobi symbol to all integers n.

primitive_root

`primitive_root P` — Smallest primitive root mod p (p prime). `undef` if none.

multiplicative_order

`multiplicative_order A, N` — smallest k > 0 with a^k ≡ 1 (mod n). `undef` if gcd(a, n) ≠ 1.

mangoldt_lambda

`mangoldt_lambda N` — von Mangoldt Λ(n) = ln(p) if n = p^k, else 0.

carmichael_lambda

`carmichael_lambda N` — Carmichael λ(n), the exponent of (Z/nZ)*.

```
p carmichael_lambda(15)   # 4
```

squares_r

`squares_r K, N` — number of representations of n as a sum of k squares (signed, ordered).

```
p squares_r(2, 5)   # 8 (e.g. ±1²+±2²)
```

thue_morse

`thue_morse N` — Thue-Morse t(n) = popcount(n) mod 2.

rudin_shapiro

`rudin_shapiro N` — Rudin-Shapiro sequence, ±1 = (-1)^a(n) where a(n) counts "11" patterns in binary(n).

farey_sequence

`farey_sequence N` — Farey sequence F_n. Returns array of `[a, b]` pairs.

```
p farey_sequence(3)   # [[0,1],[1,3],[1,2],[2,3],[1,1]]
```

frobenius_number

`frobenius_number COINS` — largest amount NOT representable as a non-negative combination of `COINS`. Closed form for two coprime denominations.

```
p frobenius_number([3, 5])   # 7
```

frobenius_solve

`frobenius_solve COINS, N` — number of ways to make change for N using `COINS`.

stern_brocot

`stern_brocot N` — Stern-Brocot tree node N (1-indexed) as `[a, b]` (Calkin-Wilf order).

Combinatorics (Wolfram Parity)

8 topics

stirling_s1

`stirling_s1 N, K` — unsigned Stirling number of the first kind |s(n,k)|.

bell_polynomial_b

`bell_polynomial_b N, K, XS` — partial Bell polynomial B_{n,k}(x_1, ..., x_{n-k+1}).

clebsch_gordan

`clebsch_gordan J1, J2, J, M1, M2 [, M]` — Clebsch-Gordan coefficient. M defaults to M1 + M2.

three_j_symbol

`three_j_symbol J1, J2, J3, M1, M2, M3` — Wigner 3-j symbol.

six_j_symbol

`six_j_symbol J1 J2 J3 J4 J5 J6` — Wigner 6-j symbol.

nine_j_symbol

`nine_j_symbol` (9 args) — Wigner 9-j as a single sum over 6-j.

debruijn_sequence

`debruijn_sequence K, N` — De Bruijn B(k, n) as an array of k^n integers.

wigner_d

`wigner_d J, M1, M2, BETA` — small Wigner d-matrix d^j_{m1, m2}(β).

q-Series, Mittag-Leffler, Coulomb

7 topics

q_pochhammer

`q_pochhammer A, Q [, N]` — (a; q)_n. Omit N for the infinite product (|q| < 1).

q_factorial

`q_factorial N, Q` — q-factorial [n]_q!.

q_binomial

`q_binomial N, K, Q` — Gaussian binomial [n choose k]_q.

q_hypergeometric_pfq

`q_hypergeometric_pfq AS, BS, Q, Z` — basic-hypergeometric series.

mittag_leffler_e

`mittag_leffler_e ALPHA, BETA, Z` — E_{α,β}(z) = Σ z^k / Γ(α k + β). Generalises exp / cosh / Mittag-Leffler.

coulomb_wave_f

`coulomb_wave_f L, ETA, RHO` — regular Coulomb wave F_L(η, ρ) via the confluent-hypergeometric representation.

coulomb_wave_g

`coulomb_wave_g L, ETA, RHO` — irregular partner G_L. Currently returns NaN; faithful Bardin/Goesnig port pending.

Inverse Special Functions

4 topics

inverse_erf

`inverse_erf Y` — inverse error function. Solves erf(x) = y.

```
p inverse_erf(0.5)   # ≈ 0.4769
```

inverse_erfc

`inverse_erfc Y` — inverse complementary error function.

inverse_gamma_regularized

`inverse_gamma_regularized A, Y` — solves P(a, x) = y for x.

inverse_beta_regularized

`inverse_beta_regularized A, B, Y` — solves I_x(a, b) = y for x via bisection.

Piecewise & Symbolic Primitives

8 topics

dirac_delta

`dirac_delta X [, EPS]` — finite-width discrete approximation: 1/eps if |x| < eps/2 else 0. EPS defaults to 1e-3.

heaviside_theta

`heaviside_theta X` — Heaviside step function: 1 (x > 0), 0 (x < 0), 0.5 (x = 0).

unit_box

`unit_box X` — 1 if |x| ? 1/2 else 0.

unit_triangle

`unit_triangle X` — max(1 − |x|, 0).

square_wave

`square_wave X [, PERIOD]` — alternates +1/-1 with the given period (default 1).

triangle_wave

`triangle_wave X [, PERIOD]` — periodic triangle wave on [-1, 1].

sawtooth_wave

`sawtooth_wave X [, PERIOD]` — periodic sawtooth on [-1, 1].

dirac_comb

`dirac_comb X, T [, EPS]` — sum of Dirac deltas at multiples of T (discrete approximation).

Number Theory (Wolfram Parity II)

14 topics

liouville_lambda

`liouville_lambda N` — Liouville λ(n) = (-1)^Ω(n).

jordan_totient

`jordan_totient K, N` — Jordan totient J_k(n) = n^k Π_{p | n} (1 - p^{-k}).

ramanujan_sum

`ramanujan_sum Q, N` — c_q(n) = Σ_{d | gcd(q,n)} μ(q/d) d.

cyclotomic_polynomial

`cyclotomic_polynomial N` — coefficient list of Φ_n(x).

```
p cyclotomic_polynomial(6)   # [1, -1, 1] = x²-x+1
```

legendre_symbol

`legendre_symbol A, P` — Legendre symbol (a/p) for prime p. -1, 0, or 1.

pythagorean_triple_q

`pythagorean_triple_q A, B, C` — 1 if $a^2 + b^2 = c^2$ (any order/sign).

gen_pythagorean_triple

`gen_pythagorean_triple M, N` — Euclid formula: returns [m^2-n^2, 2mn, m^2+n^2].

sophie_germain_q

`sophie_germain_q P` — 1 if both p and 2p+1 are prime.

mersenne_q

`mersenne_q N` — 1 if 2^n - 1 is a Mersenne prime (Lucas-Lehmer).

lucas_lehmer_test

`lucas_lehmer_test P` — Lucas-Lehmer primality test for 2^p - 1.

continued_fraction

`continued_fraction X [, N]` — first N coefficients of the simple continued-fraction expansion. N defaults to 12.

from_continued_fraction

`from_continued_fraction COEFFS` — evaluate [a0; a1, a2, ...] back to a real number.

convergents

`convergents X [, N]` — N successive [num, denom] convergents.

best_rational_approximation

`best_rational_approximation X, MAX_DENOM` — closest fraction p/q with q ⍰ MAX_DENOM. Returns [p, q].

Combinatorial Sequences

12 topics

motzkin_number

`motzkin_number N` — # paths on (0..n) using □ □ □ steps (no crossings).

narayana_number

`narayana_number N, K` — N(n, k) = (1/n) C(n,k) C(n,k-1).

delannoy_number

`delannoy_number M, N` — # paths in (m+1)×(n+1) grid using □ □ □.

schroder_number

`schroder_number N` — large Schröder number.

small_schroder_number

`small_schroder_number N` — small Schröder s_n = S_n / 2 (n □ 1).

eulerian_number

`eulerian_number N, K` — □n k□, # permutations of [n] with k ascents.

bernoulli_polynomial

`bernoulli_polynomial N, X` — B_n(x).

euler_polynomial

`euler_polynomial N, X` — E_n(x).

pell_number

`pell_number N` — Pell number P_n.

pell_lucas_number

`pell_lucas_number N` — companion Pell-Lucas Q_n.

perrin_number

`perrin_number N` — Perrin sequence P(n) = P(n-2) + P(n-3).

padovan_number

`padovan_number N` — Padovan sequence (same recurrence, different seeds).

Linear Algebra (Extended)

8 topics

kronecker_product

`kronecker_product A, B` — A ⊗ B (block-by-block outer product).

tensor_product

`tensor_product A, B` — outer product of two flat vectors → matrix A_i B_j.

tensor_contract

`tensor_contract A, B` — Frobenius inner product of two matrices.

matrix_rank

`matrix_rank A` — rank via Gaussian elimination with partial pivoting.

companion_matrix

`companion_matrix [a0, a1, …, a_{n-1}]` — companion of x^n + a_{n-1} x^{n-1} + … + a_0.

characteristic_polynomial

`characteristic_polynomial A` — coefficients [a_0, …, a_n] of det(xI − A) via Faddeev-LeVerrier.

singular_values

`singular_values A` — sorted SVDs of A (eigenvalues of A^T A, square-rooted).

nullspace

`nullspace A` — basis vectors of A's null space (zero singular values of A^T A).

Polynomial Algebra

6 topics

polynomial_gcd

`polynomial_gcd P, Q` — Euclidean GCD over real coefficients (monic).

```
p polygcd([-1, 0, 1], [-1, 1])   # x - 1
```

polynomial_quotient

`polynomial_quotient P, Q` — quotient of synthetic division.

polynomial_remainder

`polynomial_remainder P, Q` — remainder of synthetic division.

polynomial_resultant

`polynomial_resultant P, Q` — Sylvester-matrix determinant.

polynomial_discriminant

`polynomial_discriminant P` — disc(p) = (-1)^{n(n-1)/2} / a_n · res(p, p').

polynomial_roots

`polynomial_roots P` — real roots via deflation power iteration on companion matrix.

Distributions (Extended)

18 topics

gumbel_pdf

`gumbel_pdf X, MU, BETA` — Gumbel (Type I) extreme-value PDF.

gumbel_cdf

`gumbel_cdf X, MU, BETA` — Gumbel CDF.

gumbel_quantile

`gumbel_quantile P, MU, BETA` — Gumbel quantile.

frechet_pdf

`frechet_pdf X, ALPHA, S` — Fréchet (Type II) PDF.

frechet_cdf

`frechet_cdf X, ALPHA, S` — Fréchet CDF.

frechet_quantile

`frechet_quantile P, ALPHA, S` — Fréchet quantile.

logistic_pdf

`logistic_pdf X, MU, S` — Logistic PDF.

logistic_cdf

`logistic_cdf X, MU, S` — Logistic CDF.

logistic_quantile

`logistic_quantile P, MU, S` — Logistic quantile.

rayleigh_pdf

`rayleigh_pdf X, SIGMA` — Rayleigh PDF (x ? 0).

rayleigh_cdf

`rayleigh_cdf X, SIGMA` — Rayleigh CDF.

rayleigh_quantile

`rayleigh_quantile P, SIGMA` — Rayleigh quantile.

inverse_gamma_pdf

`inverse_gamma_pdf X, ALPHA, BETA` — inverse-gamma PDF.

inverse_gamma_cdf

`inverse_gamma_cdf X, ALPHA, BETA` — inverse-gamma CDF.

inverse_gamma_quantile

`inverse_gamma_quantile P, ALPHA, BETA` — inverse-gamma quantile.

kumaraswamy_pdf

`kumaraswamy_pdf X, A, B` — Kumaraswamy PDF on (0, 1).

kumaraswamy_cdf

`kumaraswamy_cdf X, A, B` — Kumaraswamy CDF.

kumaraswamy_quantile

`kumaraswamy_quantile P, A, B` — Kumaraswamy quantile.

Mathieu / Heun

4 topics

mathieu_a

`mathieu_a Q, N` — Mathieu characteristic value a_n(q) (Hill-matrix eigenvalue).

mathieu_ce

`mathieu_ce N, X, Q` — even Mathieu function ce_n(x, q) (perturbative form for small q).

mathieu_se

`mathieu_se N, X, Q` — odd Mathieu function se_n(x, q).

heun_g

`heun_g Z, A, Q, ALPHA, BETA, GAMMA, DELTA` — general Heun H_l via Frobenius series. Requires |z| < min(1, |a|).

Wavelets

4 topics

haar_transform

`haar_transform XS` — single-level Haar DWT. Returns [approx..., detail...].

haar_inverse

`haar_inverse XS` — inverse single-level Haar.

daubechies_db4

`daubechies_db4 XS` — single-level Daubechies-4 DWT (length must be even ≥ 4).

daubechies_db4_inverse

`daubechies_db4_inverse XS` — inverse single-level db4.

Graph Algorithms (Extended)

8 topics

topo_sort_adj

`topo_sort_adj ADJ` — Kahn's algorithm on adjacency-list graph. Returns ordering or `[]` on cycle. (Edge-list input → use `topological_sort` instead.)

scc_tarjan

`scc_tarjan ADJ` — Tarjan strongly-connected components. Returns array of components.

bipartite_q

`bipartite_q ADJ` — 1 if graph is 2-colourable (BFS), 0 otherwise.

max_flow_edmonds_karp

`max_flow_edmonds_karp CAP, S, T` — max flow on capacity matrix from S to T.

min_cut

`min_cut CAP, S, T` — min-cut value (= max-flow).

eccentricity

`eccentricity ADJ, V` — max BFS distance from V (unweighted).

graph_diameter

`graph_diameter ADJ` — max eccentricity over all vertices.

graph_radius

`graph_radius ADJ` — min eccentricity over all vertices.

Number-Theoretic Sums & Constants

5 topics

stieltjes_constant

`stieltjes_constant K` — kth Stieltjes constant γ_k. γ_0 = γ (Euler-Mascheroni).

gauss_sum

`gauss_sum A, P` — quadratic Gauss sum G(a, p) as `[Re, Im]`.

kloosterman_sum

`kloosterman_sum A, B, M` — K(a, b; m) as `[Re, Im]`.

eta_quotient

`eta_quotient PAIRS, Y` — Π η(d·iy)^{r_d} for [[d, r_d], ...].

root_approximant

`root_approximant X [, MAX_DENOM]` — best rational approximant (alias for `best_rational_approximation`).

Vector Calculus

6 topics

numerical_gradient

`numerical_gradient F, POINT [, H]` — central-difference ∇f. F receives a single arrayref `$_[0]`.

```
fn quad { my @y = @{$_[0]}; $y[0]**2 + 2*$y[1]**2 }
my @g = ngrad(\&quad, [3, 4])    # (6, 16)
```

numerical_jacobian

`numerical_jacobian F, POINT [, H]` — Jacobian of vector-valued F.

numerical_hessian

`numerical_hessian F, POINT [, H]` — symmetric Hessian via mixed second differences.

numerical_divergence

`numerical_divergence F, POINT [, H]` — div of vector field F.

numerical_curl

`numerical_curl F, POINT [, H]` — 3-D curl returned as `[c_x, c_y, c_z]`.

numerical_laplacian

`numerical_laplacian F, POINT [, H]` — ∇²f.

Optimization

6 topics

nelder_mead

`nelder_mead F, X0 [, MAX_ITER, TOL]` — Nelder-Mead simplex minimisation. Returns `[x*, f(x*)]`.

gradient_descent

`gradient_descent F, GRAD, X0 [, STEP, ITERS]` — fixed-step descent.

bfgs_minimize

`bfgs_minimize F, GRAD, X0 [, MAX_ITER, TOL]` — BFGS quasi-Newton with Armijo line search.

levenberg_marquardt

`levenberg_marquardt F_RESID, X0 [, MAX_ITER, TOL]` — nonlinear LSQ. F returns the residual vector.

conjugate_gradient

`conjugate_gradient A, B` — CG for symmetric positive definite A x = b.

least_squares

`least_squares A, B` — solves A^T A x = A^T b.

Numerical Integration

4 topics

romberg

`romberg F, A, B [, LEVELS]` — Romberg integration. Doubles trapezoid then Richardson-extrapolates.

gauss_legendre_quad

`gauss_legendre_quad F, A, B [, N]` — N-point Gauss-Legendre quadrature on [a, b] (default N=10).

monte_carlo_integrate

`monte_carlo_integrate F, A, B [, N]` — uniform Monte Carlo over [a, b] with N samples (default 10 000).

adaptive_simpson

`adaptive_simpson F, A, B [, TOL, MAX_DEPTH]` — adaptive Simpson with Richardson tail.

Linear Algebra (Numeric)

7 topics

lu_decompose

`lu_decompose A` → `[L, U, P]` with P A = L U; P stored as a row-permutation vector.

qr_decompose

`qr_decompose A` → `[Q, R]` via Gram-Schmidt.

householder_reflector

`householder_reflector V` → I − 2 v v^T / (v · v).

givens_rotation

`givens_rotation A, B` → `[c, s]` such that the rotation [c -s; s c] zeros the second component.

forward_substitute

`forward_substitute L, B` — solve L x = b for lower-triangular L.

back_substitute

`back_substitute U, B` — solve U x = b for upper-triangular U.

hessenberg_reduce

`hessenberg_reduce A` — orthogonal Hessenberg form via Householder reflections.

Polynomial Helpers

5 topics

poly_derivative

`poly_derivative P` — coefficient-list derivative of polynomial P (low-to-high order).

poly_integrate

`poly_integrate P [, C]` — antiderivative with constant C (default 0).

poly_compose

`poly_compose P, Q` → coefficients of P(Q(x)).

poly_eval_horner

`poly_eval_horner P, X` — Horner-rule polynomial evaluation.

pade_approximant

`pade_approximant TAYLOR, M, N` → `[num, den]` Padé approximation.

Quaternions & 3D Rotations

12 topics

quat_mul

`quat_mul A, B` — Hamilton product (w, x, y, z).

quat_conj

`quat_conj Q` — conjugate (w, -x, -y, -z).

quat_norm

`quat_norm Q` — Euclidean norm.

quat_inv

`quat_inv Q` — inverse Q* / |Q|².

quat_from_axis_angle

`quat_from_axis_angle AXIS, THETA` — unit quaternion encoding rotation about AXIS by THETA radians.

quat_to_axis_angle

`quat_to_axis_angle Q` → `[axis, theta]`.

quat_to_matrix

`quat_to_matrix Q` → 3×3 rotation matrix.

quat_from_matrix

`quat_from_matrix R` — Shepherd's algorithm.

quat_slerp

`quat_slerp A, B, T` — spherical-linear interpolation between two unit quaternions.

euler_zyx_to_matrix

`euler_zyx_to_matrix YAW, PITCH, ROLL` — intrinsic Z-Y-X Euler rotation.

matrix_to_euler_zyx

`matrix_to_euler_zyx R` → `[yaw, pitch, roll]`.

rotate_3d_vec

`rotate_3d_vec Q, V` — apply quaternion rotation to a 3-vector.

Information Theory

6 topics

kl_divergence

`kl_divergence P, Q` — D_KL(P || Q) in nats.

js_divergence

`js_divergence P, Q` — Jensen-Shannon divergence.

mutual_information

`mutual_information JOINT` — I(X; Y) from a joint-probability matrix.

cross_entropy_arr

`cross_entropy_arr P, Q` — H(P, Q) = -Σ p log q.

renyi_entropy

`renyi_entropy P [, ALPHA]` — Rényi entropy of order α (default 2).

tsallis_entropy

`tsallis_entropy P [, Q]` — Tsallis entropy of order q (default 2).

Quantum Primitives

11 topics

pauli_x

`pauli_x` — 2×2 σ_x matrix.

pauli_y

`pauli_y` — 2×2 σ_y matrix encoded with [Re, Im] entries (since stryke matrices are real).

pauli_z

`pauli_z` — 2×2 σ_z matrix.

pauli_id

`pauli_id` — 2×2 identity.

ket_bra

`ket_bra PSI, PHI` □ outer product |ψ□□φ| of two real vectors.

density_matrix

`density_matrix PSI` □ ρ = |ψ□□ψ| (real-valued).

expectation_value

`expectation_value OP, PSI` □ □ψ|O|ψ□.

commutator

`commutator A, B` — [A, B] = AB − BA.

anticommutator

`anticommutator A, B` — {A, B} = AB + BA.

partial_trace

`partial_trace RHO, DIMS, KEEP` — partial trace over subsystems whose indices are not in KEEP.

von_neumann_entropy

`von_neumann_entropy RHO` — S(ρ) = -tr(ρ log ρ) for symmetric ρ.

Statistical Mechanics

7 topics

bose_einstein

`bose_einstein E, MU, KT` — 1 / (exp((E - μ)/kT) - 1).

fermi_dirac

`fermi_dirac E, MU, KT` — 1 / (exp((E - μ)/kT) + 1).

maxwell_boltzmann_speed

`maxwell_boltzmann_speed V, M, KT` — Maxwell-Boltzmann speed PDF.

partition_function

`partition_function ENERGIES, KT` — Z = Σ exp(-E_i / kT).

helmholtz_free_energy

`helmholtz_free_energy Z, KT` — F = -kT ln Z.

boltzmann_factor

`boltzmann_factor E, KT` — exp(-E / kT).

einstein_specific_heat

`einstein_specific_heat T, THETA_E` — Einstein heat-capacity model (J / mol·K).

Optics

7 topics

fresnel_reflection_te

`fresnel_reflection_te THETA_I, N1, N2` — TE (s-polarisation) reflection coefficient.

fresnel_reflection_tm

`fresnel_reflection_tm THETA_I, N1, N2` — TM (p-polarisation) reflection.

fresnel_transmission_te

`fresnel_transmission_te THETA_I, N1, N2` — TE transmission.

fresnel_transmission_tm

`fresnel_transmission_tm THETA_I, N1, N2` — TM transmission.

abcd_thin_lens

`abcd_thin_lens F` — ABCD ray matrix for a thin lens of focal length F.

abcd_free_space

`abcd_free_space D` — ABCD matrix for free-space propagation D.

gaussian_beam_q

`gaussian_beam_q Z, W0, LAMBDA` → `[Re(q), Im(q)]` for a Gaussian beam at distance Z.

Astrodynamics

9 topics

kepler_solve

`kepler_solve M, E` — solve Kepler's equation M = E - e sin(E) by Newton iteration.

true_to_eccentric

`true_to_eccentric NU, E` → eccentric anomaly E.

eccentric_to_mean

`eccentric_to_mean E, ECC` → mean anomaly M.

julian_date

`julian_date Y, M, D, H, MN, S` — Julian date (Meeus formula).

jd_to_gregorian

`jd_to_gregorian JD` → `[Y, M, D.dd]`.

sidereal_time_gmst

`sidereal_time_gmst JD` — Greenwich Mean Sidereal Time in radians.

vis_viva

`vis_viva R, A, MU` — orbital speed from vis-viva equation.

orbital_period_kepler

`orbital_period_kepler A, MU` — $T = 2\pi \sqrt{a^3 / \mu}$.

orbital_elements_to_state

`orbital_elements_to_state A, E, I, OM, W, NU, MU` → `[r_x, r_y, r_z, v_x, v_y, v_z]`.

Time Series

4 topics

kalman_step

`kalman_step X, P, F, H, Q, R, Z` → `[X', P']` — single Kalman predict-and-update step.

exponential_smoothing

`exponential_smoothing XS, ALPHA` — first-order EMA.

holt_winters

`holt_winters XS, ALPHA, BETA, GAMMA, PERIOD` — additive triple exponential smoothing.

arma_yw_fit

`arma_yw_fit XS, P` — Yule-Walker AR(p) coefficients via Levinson-Durbin.

Graph Centrality

6 topics

pagerank

`pagerank ADJ [, DAMPING, ITERS]` — PageRank via power iteration. Default damping 0.85, 100 iters.

betweenness_centrality

`betweenness_centrality ADJ` — Brandes' algorithm (unweighted).

closeness_centrality

`closeness_centrality ADJ` — closeness centrality. 0 for disconnected vertices.

eigenvector_centrality

`eigenvector_centrality ADJ` — power iteration on the adjacency matrix.

degree_centrality

`degree_centrality ADJ` — degree / (n - 1).

triangle_count

`triangle_count ADJ` — number of triangles incident at each vertex.

Random Samplers (Extended)

6 topics

rgumbel

`rgumbel MU, BETA` — Gumbel sample.

rfrechet

`rfrechet ALPHA, S` — Fréchet sample.

rrayleigh

`rrayleigh SIGMA` — Rayleigh sample.

rlogistic

`rlogistic MU, S` — logistic sample.

rkumaraswamy

`rkumaraswamy A, B` — Kumaraswamy sample on (0, 1).

rinverse_gamma

`rinverse_gamma ALPHA, BETA` — inverse-gamma sample (Marsaglia-Tsang).

2-D Geometry

3 topics

graham_scan

`graham_scan POINTS` — 2-D convex hull (CCW).

line_line_intersect_2d

`line_line_intersect_2d P1, P2, P3, P4` ⍰ intersection `[x, y]` or undef when parallel.

point_segment_distance

`point_segment_distance P, A, B` — Euclidean distance from P to segment AB.

Auto-Differentiation

2 topics

forward_diff

`forward_diff F, X` — derivative of scalar f at scalar x (Julia ForwardDiff style; central-difference fallback).

forward_diff_grad

`forward_diff_grad F, X_VEC` — gradient at vector x; alias for `numerical_gradient`.

Statistical Tests (R parity)

12 topics

bartlett_test

`bartlett_test GROUPS` — Bartlett's k-sample variance equality test. Returns `[chi², df, p]`.

levene_test

`levene_test GROUPS` — Levene's variance-equality test. Returns `[F, df1, df2, p]`.

fishers_exact_test_2x2

`fishers_exact_test_2x2 [[a, b], [c, d]]` — two-sided p-value.

mcnemar_test

`mcnemar_test [[a, b], [c, d]]` — paired 2×2 table with continuity correction. Returns `[chi², p]`.

runs_test

`runs_test SEQ` — Wald-Wolfowitz runs test on a binary sequence. Returns `[Z, p]`.

friedman_test

`friedman_test BLOCKS_BY_TREATMENTS` — non-parametric ANOVA on ranks. Returns `[Q, df, p]`.

kruskal_wallis_test

`kruskal_wallis_test GROUPS` — non-parametric ANOVA. Returns `[H, df, p]`.

sign_test

`sign_test PAIRS` — paired sign test. Returns `[N+, p]`.

anderson_darling_normality

`anderson_darling_normality XS` — A^2 statistic. > 0.752 ⍰ reject normality at α=0.05.

jarque_bera_test

`jarque_bera_test XS` — normality test from sample skewness + kurtosis.

ljung_box_test

`ljung_box_test XS [, H]` — autocorrelation test. Returns `[Q, h, p]`.

durbin_watson_stat

`durbin_watson_stat RESID` — DW autocorrelation statistic on residuals.

Distance Metrics (Distances.jl parity)

6 topics

mahalanobis_distance

`mahalanobis_distance X, MU, SIGMA` — √((x-μ)ᵀ Σ⁻¹ (x-μ)).

cosine_distance

`cosine_distance A, B` — 1 − cos(A, B).

canberra_distance

`canberra_distance A, B` — Σ |a-b| / (|a|+|b|).

bray_curtis_distance

`bray_curtis_distance A, B` — Σ|a-b| / Σ(a+b).

l1_distance

`l1_distance A, B` — Manhattan / taxi-cab distance.

chi_squared_distance

`chi_squared_distance A, B` — ½ Σ (a-b)² / (a+b).

Multivariate / Non-Central Distributions

9 topics

multivariate_normal_pdf

`multivariate_normal_pdf X, MU, SIGMA` — multivariate Gaussian PDF.

multivariate_normal_sample

`multivariate_normal_sample MU, SIGMA` — sample from MVN via Cholesky.

dirichlet_pdf

`dirichlet_pdf X, ALPHA` — Dirichlet PDF on the simplex.

dirichlet_sample

`dirichlet_sample ALPHA` — sample via independent Gamma draws.

skellam_pmf

`skellam_pmf K, MU1, MU2` — difference of two independent Poisson(μ) variates.

inverse_gaussian_pdf

`inverse_gaussian_pdf X, MU, LAMBDA` — Wald PDF.

inverse_gaussian_cdf

`inverse_gaussian_cdf X, MU, LAMBDA` — Wald CDF.

inverse_gaussian_sample

`inverse_gaussian_sample MU, LAMBDA` — Michael-Schucany-Haas.

non_central_chi2_pdf

`non_central_chi2_pdf X, K, LAMBDA` — Series-form non-central χ^2 PDF.

Matrix Functions

5 topics

matrix_exp

`matrix_exp A` — e^A via 13-term Padé + scaling-and-squaring.

matrix_log

`matrix_log A` — ln A via inverse scaling (Denman-Beavers SQRT) + Padé log.

matrix_sqrt

`matrix_sqrt A` — A^(1/2) via Denman-Beavers iteration.

matrix_sin

`matrix_sin A` — series form Σ (-1)^k A^(2k+1) / (2k+1)!.

matrix_cos

`matrix_cos A` — series form Σ (-1)^k A^(2k) / (2k)!.

Adaptive ODE Solvers

4 topics

rk45_dormand_prince

`rk45_dormand_prince F, Y0, T0, T1, H0 [, TOL]` — adaptive Dormand-Prince. F receives `[t, y0, …]`.

midpoint_step

`midpoint_step F, Y, T, H` — single explicit-midpoint step.

heun_step

`heun_step F, Y, T, H` — improved-Euler / Heun step.

verlet_step

`verlet_step ACCEL, Q, P, T, H` — velocity-Verlet symplectic step.

Generalized Linear Models

4 topics

logistic_regression

`logistic_regression X, Y` — IRLS (no intercept added — supply column of ones).

poisson_regression

`poisson_regression X, Y` — IRLS with log link.

ridge_regression

`ridge_regression X, Y [, LAMBDA]` — Tikhonov-regularised LSQ.

lasso_coord

`lasso_coord X, Y [, LAMBDA, MAX_ITER]` — coordinate-descent LASSO.

Bootstrap & Resampling

3 topics

bootstrap_mean_ci

`bootstrap_mean_ci DATA [, B, ALPHA]` — percentile CI for the mean.

jackknife_estimate

`jackknife_estimate DATA` — leave-one-out SE of the mean.

permutation_test_diff

`permutation_test_diff A, B [, N_ITER]` — two-sided p of mean difference.

Time Series Operators

4 topics

acf_at_lag

`acf_at_lag XS, LAG` — autocorrelation at one lag.

diff_op

`diff_op XS [, LAG]` — y_t – y_{t-lag}.

lag_op

`lag_op XS [, K]` — shift by k (NaN-pad).

decompose_classical

`decompose_classical XS [, PERIOD]` — additive [trend, season, residual].

Combinatorial Generators

4 topics

combinations_list

`combinations_list N, K` — all k-subsets of 0..n in lex order.

permutations_list

`permutations_list N` — all permutations of 0..n via Heap's algorithm.

cyclic_permutations

`cyclic_permutations ARR` — all rotations.

subsets_of_size

`subsets_of_size ARR, K` — all k-subsets of an arbitrary array.

Dynamic Programming

5 topics

longest_increasing_subseq

`longest_increasing_subseq XS` — length only (O(n log n)).

knapsack_01

`knapsack_01 ITEMS, CAPACITY` — items as `[[w, v], …]`.

subset_sum_target

`subset_sum_target ARR, T` — 1 if any subset sums to T.

coin_change_min

`coin_change_min COINS, N` — min number of coins to make N (-1 if impossible).

edit_distance_levenshtein

`edit_distance A, B` — Levenshtein distance.

ML Metrics

5 topics

one_hot_encode

`one_hot_encode LABELS [, N_CLASSES]` → n×K matrix.

label_encode

`label_encode ARR` → `[indices, classes]`.

categorical_cross_entropy

`categorical_cross_entropy Y_TRUE, Y_PRED` — averaged over batch.

classification_metrics

`classification_metrics TP, FP, FN, TN` → `[acc, prec, recall, f1]`.

roc_auc

`roc_auc SCORES, LABELS` — Mann-Whitney U / (n+ n−).

DSP / Image Filters

6 topics

gaussian_blur_kernel

`gaussian_blur_kernel SIGMA [, RADIUS]` — 1-D normalized kernel.

sobel_x

`sobel_x` — horizontal Sobel kernel.

sobel_y

`sobel_y` — vertical Sobel kernel.

prewitt_x

`prewitt_x` — horizontal Prewitt.

prewitt_y

`prewitt_y` — vertical Prewitt.

laplacian_of_gaussian

`laplacian_of_gaussian SIGMA` — 2-D LoG kernel.

Stochastic Processes

4 topics

brownian_path

`brownian_path T, N` — discrete Wiener path on [0, T].

geometric_brownian_path

`geometric_brownian_path S0, MU, SIGMA, T, N` — GBM sample path.

poisson_process

`poisson_process LAMBDA, T` — homogeneous Poisson arrival times.

random_walk_1d

`random_walk_1d N [, P]` — ±1 walk over N steps.

Compression / Information Complexity

3 topics

lempel_ziv_complexity

`lempel_ziv_complexity SEQ` — LZ76 production count.

huffman_code_lengths

`huffman_code_lengths FREQS` — code-length per symbol.

shannon_entropy_rate

`shannon_entropy_rate SEQ [, M]` — H_m − H_{m-1} block estimate.

Physics & Quantum (Extended)

5 topics

planck_blackbody

`planck_blackbody LAMBDA, T` — spectral radiance B(λ, T).

rayleigh_jeans

`rayleigh_jeans LAMBDA, T` — long-wavelength approx.

compton_shift

`compton_shift THETA` — photon wavelength shift Δλ = (h/m_e c)(1-cosθ).

rydberg_energy

`rydberg_energy N` — hydrogen level energy −13.605693/n² eV.

hydrogen_radial_wavefunction

`hydrogen_radial_wavefunction N, L, R` — atomic-units R_{n,l}(r).

Number Theory & Algebra (Extended)

5 topics

integer_log

`integer_log N [, BASE]` — largest k with base^k ≤ n.

aks_primality

`aks_primality N` — deterministic primality (perfect-power + small-factor + Miller-Rabin fallback).

elliptic_curve_add

`elliptic_curve_add P, Q, A, B` — group law on $y^2 = x^3 + a\,x + b$. Identity = `[NaN, NaN]`.

berlekamp_massey

`berlekamp_massey SEQ` — minimal-LFSR connection-polynomial coefficients.

bezout_coefficients

`bezout_coefficients A, B` → `[gcd, x, y]` with $a\,x + b\,y = \gcd$.

CAS-Lite

3 topics

factor_quadratic

`factor_quadratic A, B, C` — real roots of $a\,x^2 + b\,x + c = 0$.

complete_square

`complete_square A, B, C [h, k]` for $a\,(x-h)^2 + k$.

partial_fraction_simple

`partial_fraction_simple NUM, ROOTS` — Heaviside cover-up residues.

Quadrature (Specialised)

7 topics

gauss_chebyshev_quad

`gauss_chebyshev_quad F [, N]` — □_{-1}^1 f(x)/□(1-x²) dx (default N=16).

gauss_hermite_quad

`gauss_hermite_quad F [, N]` — □_{-□}^□ f(x) e^{-x²} dx (default N=20).

gauss_laguerre_quad

`gauss_laguerre_quad F [, N]` — □_0^□ f(x) e^{-x} dx (default N=20).

clenshaw_curtis_quad

`clenshaw_curtis_quad F, A, B [, N]` — Chebyshev-node quadrature.

tanh_sinh_quad

`tanh_sinh_quad F, A, B [, LEVELS]` — double-exponential quadrature.

gauss_legendre_2d

`gauss_legendre_2d F, AX, BX, AY, BY [, N]` — Cartesian-product GL.

monte_carlo_2d

`monte_carlo_2d F, AX, BX, AY, BY [, N]` — uniform Monte Carlo on a rectangle.

Optimization (Heuristic)

3 topics

simulated_annealing

`simulated_annealing F, X0 [, T0, COOL, ITERS, STEP]` — Metropolis-Hastings minimisation.

simplex_lp

`simplex_lp C, A, B` — max c⋅x s.t. Ax ≤ b, x ≥ 0. Returns `[x, value]`.

particle_swarm

`particle_swarm F, BOUNDS, SWARM, ITERS` — constriction-coefficient PSO.

Distributions (Tail / Mixture / Categorical)

14 topics

gev_pdf

`gev_pdf X, MU, SIGMA, XI` — Generalised Extreme Value PDF.

gev_cdf

`gev_cdf X, MU, SIGMA, XI` — GEV CDF.

gev_sample

`gev_sample MU, SIGMA, XI` — GEV sample.

gen_pareto_pdf

`gen_pareto_pdf X, MU, SIGMA, XI` — Generalised Pareto PDF.

gen_pareto_cdf

`gen_pareto_cdf X, MU, SIGMA, XI` — GP CDF.

gen_pareto_sample

`gen_pareto_sample MU, SIGMA, XI` — GP sample.

skew_normal_pdf

`skew_normal_pdf X, XI, OMEGA, ALPHA` — Azzalini skew-normal PDF.

skew_normal_cdf

`skew_normal_cdf X, XI, OMEGA, ALPHA` — skew-normal CDF (numeric integration).

mixture_normal_pdf

`mixture_normal_pdf X, WEIGHTS, MEANS, STDS` — Gaussian-mixture PDF.

categorical_sample

`categorical_sample PROBS` — sample category index.

multinomial_pmf

`multinomial_pmf COUNTS, PROBS` — multinomial PMF.

multinomial_sample

`multinomial_sample N, PROBS` — multinomial counts vector.

truncated_normal_pdf

`truncated_normal_pdf X, MU, SIGMA, LO, HI` — truncated-normal PDF.

truncated_normal_sample

`truncated_normal_sample MU, SIGMA, LO, HI` — rejection sampler.

Clustering & Dimensionality Reduction

7 topics

dbscan

`dbscan POINTS, EPS, MIN_PTS` — density-based clustering. Returns label per point (-1 = noise).

gmm_em_1d

`gmm_em_1d DATA, K [, MAX_ITER]` — 1-D Gaussian-mixture EM. Returns `[π, μ, Ω]`.

silhouette_score

`silhouette_score POINTS, LABELS` — mean silhouette coefficient.

davies_bouldin_index

`davies_bouldin_index POINTS, LABELS` — lower is better.

calinski_harabasz_index

`calinski_harabasz_index POINTS, LABELS` — higher is better.

mds_2d

`mds_2d D` — classical multidimensional scaling on a distance matrix → 2-D coords.

mean_shift

`mean_shift POINTS [, BANDWIDTH, MAX_ITER]` — non-parametric mode-seeking.

Neural-Net Primitives

11 topics

batch_norm

`batch_norm XS [, EPS]` — zero-mean unit-variance normalisation.

layer_norm

`layer_norm XS [, EPS]` — alias for batch_norm (per-sample reuse).

dropout_mask

`dropout_mask N [, P]` — Bernoulli dropout mask scaled by 1/(1−p).

max_pool_1d

`max_pool_1d XS [, WIN]` — non-overlapping max-pool.

avg_pool_1d

`avg_pool_1d XS [, WIN]` — non-overlapping average-pool.

attention_softmax

`attention_softmax XS` — numerically stable softmax.

positional_encoding

`positional_encoding LENGTH, D_MODEL` — sinusoidal Transformer encoding.

glorot_init

`glorot_init FAN_IN, FAN_OUT` — Glorot uniform.

he_init

`he_init FAN_IN, FAN_OUT` — Gaussian (std = √(2/fan_in)).

adam_step

`adam_step PARAM, GRAD, M, V, LR, B1, B2, EPS, T` → `[param', m', v']`.

rmsprop_step

`rmsprop_step PARAM, GRAD, V, LR, DECAY, EPS` → `[param', v']`.

Time Series (Spectral)

5 topics

ewma

`ewma XS [, ALPHA]` — exponentially-weighted moving average.

ccf

`ccf XS, YS [, MAX_LAG]` — sample cross-correlation function over -L..L lags.

periodogram

`periodogram XS` — squared-magnitude DFT / N (one-sided spectrum).

welch_psd

`welch_psd XS [, SEG_LEN, OVERLAP]` — Welch's PSD via Hann segments.

lag_features

`lag_features XS, P` — design matrix of P lagged features.

Image Processing

5 topics

median_filter_2d

`median_filter_2d IMG [, RADIUS]` — denoising median filter.

threshold_otsu

`threshold_otsu IMG` — Otsu's optimal threshold for 0..255 grayscale.

histogram_equalize

`histogram_equalize IMG` — global histogram equalisation.

erode_2d

`erode_2d IMG [, RADIUS]` — morphological erosion (square structuring element).

dilate_2d

`dilate_2d IMG [, RADIUS]` — morphological dilation.

Loss Functions

3 topics

mse_loss

`mse_loss A, B` — mean-squared error.

mae_loss

`mae_loss A, B` — mean-absolute error.

huber_loss

`huber_loss A, B [, DELTA]` — Huber loss (default δ=1).

Spatial / Geographic

3 topics

vincenty_distance

`vincenty_distance LAT1, LON1, LAT2, LON2` — geodesic distance on WGS-84.

mercator_project

`mercator_project LAT, LON [, R]` → `[x, y]` Mercator metres.

destination_from_bearing

`destination_from_bearing LAT, LON, BEARING_DEG, DIST_M [, R]` — spherical great-circle target.

Integer Sequences

6 topics

recaman

`recaman N` — first N terms of Recamán's sequence.

sylvester

`sylvester N` — Sylvester's sequence (a_n = a_{n-1}² − a_{n-1} + 1).

happy_q

`happy_q N` — 1 if N is a happy number.

amicable_pair_q

`amicable_pair_q A, B` — 1 if (a, b) is an amicable pair.

aliquot_sequence

`aliquot_sequence N [, MAX_STEPS]` — sum-of-proper-divisors trajectory.

magic_constant

`magic_constant N` — n(n²+1)/2 (sum of any line of an n×n magic square).

Graph Link & Cluster Metrics

7 topics

clustering_coefficient_local

`clustering_coefficient_local ADJ` — local CC per vertex.

clustering_coefficient_global

`clustering_coefficient_global ADJ` — mean local CC.

assortativity

`assortativity ADJ` — degree-degree correlation coefficient.

common_neighbors

`common_neighbors ADJ, U, V` — count.

jaccard_neighbors

`jaccard_neighbors ADJ, U, V` — |Γ(u) ∩ Γ(v)| / |Γ(u) ∪ Γ(v)|.

adamic_adar

`adamic_adar ADJ, U, V` — Σ 1/log(deg(w)) over common neighbours w.

preferential_attachment_score

`preferential_attachment_score ADJ, U, V` — deg(u)·deg(v).

3-D Geometry

8 topics

triangle_3d_normal

`triangle_3d_normal P1, P2, P3` — unit normal.

triangle_3d_area

`triangle_3d_area P1, P2, P3` — area via cross product.

tetrahedron_volume

`tetrahedron_volume A, B, C, D` — |det(B-A, C-A, D-A)|/6.

plane_from_3_points

`plane_from_3_points P1, P2, P3` → `[a, b, c, d]` (a x + b y + c z + d = 0).

point_to_plane_distance

`point_to_plane_distance P, [a, b, c, d]`.

ray_triangle_intersect

`ray_triangle_intersect O, D, A, B, C` — Möller-Trumbore. Returns t or NaN.

ray_sphere_intersect

`ray_sphere_intersect O, D, CENTER, RADIUS` — nearest non-negative t.

aabb_overlap

`aabb_overlap A_MIN, A_MAX, B_MIN, B_MAX` — 1 if 3-D AABBs overlap.

Iterative Solvers

6 topics

gauss_seidel

`gauss_seidel A, B [, MAX_ITER, TOL]` — Gauss-Seidel iterative solver.

jacobi_iteration

`jacobi_iteration A, B [, MAX_ITER, TOL]` — Jacobi iteration.

sor_solve

`sor_solve A, B [, OMEGA, MAX_ITER, TOL]` — Successive Over-Relaxation.

thomas_tridiag_solve

`thomas_tridiag_solve A_SUB, B_MAIN, C_SUP, D` — tridiagonal direct solve.

richardson_extrapolation

`richardson_extrapolation F, H0 [, LEVELS]` — Romberg-style extrapolation.

finite_difference_5pt

`finite_difference_5pt F, X [, H]` — five-point central derivative.

Crypto / Modular Algebra

5 topics

tonelli_shanks_sqrt

`tonelli_shanks_sqrt N, P` — modular square root mod prime p.

baby_step_giant_step

`baby_step_giant_step G, H, P` — discrete logarithm mod p.

pollard_rho_factor

`pollard_rho_factor N` — return a non-trivial factor of N.

modular_lcm

`modular_lcm XS` — LCM via gcd.

crt_general

`crt_general REMAINDERS, MODULI` — CRT over arbitrary (not coprime) moduli; -1 if infeasible.

Physics & Chemistry

5 topics

van_der_waals_p

`van_der_waals_p N, T, V, A, B` — pressure from Van der Waals equation.

nernst_equation

`nernst_equation E0, N, T, Q` — half-cell potential.

arrhenius_rate

`arrhenius_rate A, EA, T` — k = A exp(-Ea / RT).

reduced_mass

`reduced_mass M1, M2` — μ = m1 m2 / (m1 + m2).

ph_to_concentration

`ph_to_concentration PH` — [H⁺] = 10^{-pH}.

MCMC & SDEs

5 topics

metropolis_hastings

`metropolis_hastings LOG_PI, X0, SIGMA, ITERS [, BURN_IN]` — random-walk MH.

gibbs_sampler_step

`gibbs_sampler_step COND_SAMPLERS, X` — one Gibbs sweep over coordinates.

euler_maruyama

`euler_maruyama MU, SIGMA, X0, T0, T_END, N_STEPS` — Euler-Maruyama SDE integrator.

milstein

`milstein MU, SIGMA, SIGMA_X, X0, T0, T_END, N_STEPS` — Milstein scheme.

ornstein_uhlenbeck_path

`ornstein_uhlenbeck_path THETA, MU, SIGMA, X0, T, N` — exact OU sample path.

Hidden Markov Models

3 topics

hmm_forward

`hmm_forward PI, A, B, OBS` — log P(O | λ).

hmm_viterbi

`hmm_viterbi PI, A, B, OBS` — most-likely state path.

hmm_backward

`hmm_backward PI, A, B, OBS` — backward β matrix.

Survival Analysis

2 topics

kaplan_meier

`kaplan_meier TIMES, EVENTS` → matrix of [t, S(t)].

log_rank_test

`log_rank_test T1, E1, T2, E2` → `[chi², p]`.

Sequence Alignment

2 topics

needleman_wunsch

`needleman_wunsch A, B [, MATCH, MISMATCH, GAP]` — global alignment score.

smith_waterman

`smith_waterman A, B [, MATCH, MISMATCH, GAP]` — best local-alignment score.

Chemistry

4 topics

gibbs_free_energy

`gibbs_free_energy DH [, T, DS]` — $\Delta G = \Delta H - T\Delta S$.

henderson_hasselbalch

`henderson_hasselbalch PKA, A_CONJ, HA` — $pH = pKa + \log10([A^-]/[HA])$.

radioactive_decay

`radioactive_decay N0, LAMBDA, T` — $N(t) = N_0$ e^{-λt}.

half_life_to_constant

`half_life_to_constant T_HALF` — $\lambda = \ln 2$ / t_½.

Control Theory

5 topics

pid_step

`pid_step STATE, KP, KI, KD, E, DT` → `[new_state, u]` (PID controller).

transfer_function_eval

`transfer_function_eval NUM, DEN, OMEGA` → `[Re, Im]` of H(jω).

bode_magnitude_db

`bode_magnitude_db NUM, DEN, OMEGA` — $20 \log_{10} |H(j\omega)|$.

bode_phase_deg

`bode_phase_deg NUM, DEN, OMEGA` — phase of H(jω) in degrees.

lqr_2x2

`lqr_2x2 A, B, Q, R` — closed-form 2×2 continuous LQR. Returns `[K, P]`.

Game Theory

3 topics

nash_eq_2x2

`nash_eq_2x2 P1, P2` — pure Nash equilibria of a 2×2 bimatrix game.

shapley_value

`shapley_value N, V_TABLE` — Shapley values from characteristic-function vector.

expected_utility

`expected_utility PROBS, PAYOFFS` — Σ p·u.

Operations Research

3 topics

hungarian_assignment

`hungarian_assignment COST_MATRIX` — assignment + total cost. (Greedy heuristic; reduces to Hungarian for small instances.)

tsp_nearest_neighbor

`tsp_nearest_neighbor DIST_MATRIX [tour, length]`.

vertex_cover_2approx

`vertex_cover_2approx ADJ` — 2-approximation greedy.

Numerical PDE

3 topics

heat_eq_1d

`heat_eq_1d F, A, B, NX, T, NT, ALPHA` — explicit FTCS heat-equation solver.

wave_eq_1d

`wave_eq_1d F, A, B, NX, T, NT, C` — explicit wave-equation solver.

laplace_2d_jacobi

`laplace_2d_jacobi GRID [, MAX_ITER, TOL]` — Jacobi smoother on a Dirichlet grid.

Bayesian Conjugate Updates

4 topics

beta_binomial_update

`beta_binomial_update ALPHA, BETA, N, K` → posterior `[α', β']`.

normal_normal_update

`normal_normal_update MU0, VAR0, N, YBAR, VAR_DATA` → `[μ_post, σ²_post]`.

gamma_poisson_update

`gamma_poisson_update ALPHA, BETA, N, TOTAL_EVENTS` → `[α', β']`.

dirichlet_multinomial_update

`dirichlet_multinomial_update ALPHA, COUNTS` → $\alpha + \text{counts}$.

Quantum Computing Gates

8 topics

hadamard_gate

`hadamard_gate` — 2×2 Hadamard.

cnot_gate

`cnot_gate` — 4×4 controlled-NOT.

swap_gate

`swap_gate` — 4×4 SWAP.

cz_gate

`cz_gate` — 4×4 controlled-Z.

qft_matrix

`qft_matrix N` → `[Re, Im]` of N×N QFT matrix.

phase_gate

`phase_gate PHI` → `[Re, Im]` of diag(1, e^iφ).

s_gate

`s_gate` — phase(π/2).

t_gate

`t_gate` — phase(π/4).

Splines & Curves

4 topics

bezier_eval

`bezier_eval CONTROL_PTS, T` — de Casteljau evaluation.

catmull_rom_eval

`catmull_rom_eval P0, P1, P2, P3, T` — uniform Catmull-Rom.

cubic_hermite_eval

`cubic_hermite_eval P0, M0, P1, M1, T` — Hermite basis interpolation.

bspline_basis

`bspline_basis I, K, T, KNOTS` — Cox-de Boor B-spline basis.

Music & Audio

4 topics

freq_to_midi

`freq_to_midi F` — MIDI note number from frequency.

midi_to_freq

`midi_to_freq M` — frequency from MIDI note.

equal_temperament_freq

`equal_temperament_freq SEMITONES_ABOVE_A4` — 12-TET frequency.

cents_difference

`cents_difference F1, F2` — 1200 $\log_2$(F2/F1).

Astronomy & Cosmology

3 topics

redshift_z

`redshift_z LAMBDA_OBS, LAMBDA_EMIT` — z = λ_obs/λ_emit − 1.

hubble_distance

`hubble_distance H0` — c / H_0 in Mpc.

luminosity_distance

`luminosity_distance Z, H0, OMEGA_M, OMEGA_L` — flat ΛCDM (numeric).

Fluid Dynamics

4 topics

reynolds_number

`reynolds_number RHO, U, L, MU` — ρ U L / μ.

mach_number

`mach_number U, C` — U / c.

prandtl_number

`prandtl_number CP, MU, K` — c_p μ / k.

bernoulli_velocity

`bernoulli_velocity P1, P2, RHO` — √(2(p₁ − p₂)/ρ).

Distributions (Discrete / Circular)

4 topics

negative_binomial_pmf

`negative_binomial_pmf K, R, P` — failures-before-rth-success PMF.

hypergeometric_pmf

`hypergeometric_pmf N, K_POP, N_DRAWS, K_OBS` — finite-population PMF.

beta_binomial_pmf

`beta_binomial_pmf K, N, ALPHA, BETA` — beta-binomial PMF.

von_mises_pdf

`von_mises_pdf THETA, MU, KAPPA` — circular Gaussian.

Random Graph Generators

3 topics

erdos_renyi_random

`erdos_renyi_random N, P` — G(n, p) Bernoulli edge model.

barabasi_albert_random

`barabasi_albert_random N, M` — preferential-attachment model.

watts_strogatz_random

`watts_strogatz_random N, K, P` — small-world rewiring.

Color Science

3 topics

rgb_to_lab

`rgb_to_lab R, G, B` — sRGB → CIE Lab.

lab_to_rgb

`lab_to_rgb L, A, B` — CIE Lab → sRGB.

kelvin_to_rgb

`kelvin_to_rgb K` — black-body sRGB approximation (Tanner Helland).

Integer Sequences (Combinatorial)

4 topics

bell_triangle

`bell_triangle N` — first N+1 rows of the Bell / Aitken triangle.

surjection_count

`surjection_count N, K` — k! · S(n, k).

distinct_partition_count

`distinct_partition_count N` — partitions of N into distinct parts.

fibonacci_q

`fibonacci_q N` — 1 if N is a Fibonacci number.

Multiple-Testing Corrections

3 topics

bonferroni_correction

`bonferroni_correction PVALS` — Bonferroni-adjusted p-values (capped at 1).

benjamini_hochberg

`benjamini_hochberg PVALS` — BH FDR q-values.

tukey_hsd

`tukey_hsd ALPHA, K, DF, MSE, N` — Tukey honestly-significant-difference critical bound.

Probabilistic Distances

3 topics

hellinger_distance

`hellinger_distance P, Q` — √(½ Σ (√p − √q)²).

wasserstein_1d

`wasserstein_1d A, B` — earth-mover distance for sorted samples.

chi_squared_divergence

`chi_squared_divergence P, Q` — Σ (p − q)² / q.

Distributions (Tail / Inflated)

3 topics

beta_geometric_pmf

`beta_geometric_pmf K, ALPHA, BETA` — beta-geometric PMF.

generalized_gamma_pdf

`generalized_gamma_pdf X, A, D, P` — generalised-gamma PDF.

zip_pmf

`zip_pmf K, PI, LAMBDA` — zero-inflated Poisson.

Astrophysics & Radiation

9 topics

stefan_boltzmann_luminosity

`stefan_boltzmann_luminosity R, T` — $L = 4\pi R^2 \sigma T^4$.

photon_momentum

`photon_momentum LAMBDA` — h / λ.

photon_energy_ev

`photon_energy_ev LAMBDA` — hc / λ in eV.

dipole_radiation_power

`dipole_radiation_power Q, A` — Larmor formula.

parallax_to_distance

`parallax_to_distance P_ARCSEC` — parsecs from arcseconds.

hawking_temperature

`hawking_temperature M_KG` — black-hole temperature.

roche_limit

`roche_limit R, RHO_PRIMARY, RHO_SAT` — rigid-body Roche limit.

apparent_magnitude

`apparent_magnitude ABS_M, D_PC` — $m = M + 5 \log_{10}(d/10)$.

distance_modulus

`distance_modulus D_PC` — $\mu = 5 \log d - 5$.

Chemistry (Beer / Rate / Colligative)

3 topics

beer_lambert

`beer_lambert EPSILON, L, C` — A = ε·l·c.

rate_law_n

`rate_law_n A0, K, T, N` — concentration after time t for nth-order reaction.

freezing_point_depression

`freezing_point_depression KF, MOLALITY, I` — ΔT = K_f m i.

Mixed-Strategy Game Theory

2 topics

`mixed_nash_2x2`

`mixed_nash_2x2 P1, P2` → `[p, q]` mixed-strategy probabilities.

`minimax_2x2`

`minimax_2x2 M` — security level for the row player.

Computer Graphics

4 topics

barycentric_coords_2d

`barycentric_coords_2d P, A, B, C` → `[u, v, w]`.

bresenham_line

`bresenham_line X0, Y0, X1, Y1` → array of integer pixels.

bilinear_interp_2d

`bilinear_interp_2d F00, F10, F01, F11, U, V` — unit-square bilinear.

point_in_polygon_2d

`point_in_polygon_2d P, POLY` — ray-casting test.

DSP (Hilbert / Cepstrum / Filter Design)

5 topics

hilbert_transform

`hilbert_transform XS` — discrete Hilbert via DFT.

cepstrum

`cepstrum XS` — real cepstrum ifft(log|fft|).

butterworth_lowpass_coeffs

`butterworth_lowpass_coeffs ORDER, CUTOFF` → `[b, a]` digital biquad-cascade.

savitzky_golay_coeffs

`savitzky_golay_coeffs NL, NR, M` — SG filter coefficients.

savitzky_golay_filter

`savitzky_golay_filter XS, COEFFS` — apply pre-computed SG coefficients.

Image Processing (Edges / Bilateral)

2 topics

canny_edge_intensity

`canny_edge_intensity IMG` — Sobel-magnitude edge map (caller thresholds).

bilateral_filter_basic

`bilateral_filter_basic IMG, RADIUS, SIGMA_S, SIGMA_R` — edge-preserving smoother.

Clustering Helpers

2 topics

kmeans_pp_init

`kmeans_pp_init POINTS, K` — k-means++ seed centroids.

elbow_score

`elbow_score POINTS, LABELS` — within-cluster sum of squares.

Combinatorics (Tableaux / Boustrophedon)

4 topics

young_tableaux_count

`young_tableaux_count LAMBDA` — hook-length formula.

euler_alt_permutation

`euler_alt_permutation N` — Euler / zigzag number A000111.

genocchi_number

`genocchi_number N` — G_n via Bernoulli identity.

lattice_paths_count

`lattice_paths_count M, N` — C(m+n, n) east/north paths.

Number Theory (Tetration / Smoothness)

4 topics

tetration

`tetration A, N` — N-fold a-tower (overflow □□).

ackermann_limited

`ackermann_limited M, N` — depth-limited Ackermann (refuses past M=3 N=14).

perfect_power_q

`perfect_power_q N` — 1 if N = a^b for integers a, b □ 2.

b_smooth_q

`b_smooth_q N, B` — 1 if every prime factor of N is □ B.

Network Coreness & Structure

2 topics

k_core

`k_core ADJ` — vertex coreness.

rich_club_coefficient

`rich_club_coefficient ADJ, K` — fraction of edges among nodes with deg > k.

Crypto Helpers

3 topics

rsa_basic_encrypt

`rsa_basic_encrypt M, E, N` — m^e mod n.

rsa_basic_decrypt

`rsa_basic_decrypt C, D, N` — c^d mod n.

dh_shared_secret

`dh_shared_secret PEER_PUB, PRIVATE, P` — Diffie-Hellman.

Quantum Entanglement

4 topics

bell_state_phi_plus

`bell_state_phi_plus` — Bell state (|00⟩+|11⟩)/√2.

bell_state_psi_minus

`bell_state_psi_minus` — singlet (|01⟩-|10⟩)/√2.

density_matrix_purity

`density_matrix_purity RHO` — tr(ρ²).

concurrence_2qubit

`concurrence_2qubit PSI` — 2|ad − bc| in real basis.

2-D Geometry (Polygons & Circles)

4 topics

point_in_circle

`point_in_circle P, CENTER, R` — 1 if inside.

circle_circle_intersect_2d

`circle_circle_intersect_2d C1, R1, C2, R2` — 0/1/2 intersection points.

polygon_centroid

`polygon_centroid POLY` — area-weighted centroid.

sutherland_hodgman_clip

`sutherland_hodgman_clip SUBJECT, CLIP` — clip against convex CCW polygon.

Time-Series Smoothing

1 topics

kalman_rts_smoother

`kalman_rts_smoother X_HAT, P, X_PRED, P_PRED, F` — RTS smoothed means.

Bioinformatics

6 topics

gc_content

`gc_content SEQ` — fraction of G + C in DNA / RNA string.

codon_to_aa

`codon_to_aa CODON` — single-letter amino-acid (or '*' / 'X').

reverse_complement_dna

`reverse_complement_dna SEQ` — DNA reverse complement.

hamming_dna

`hamming_dna A, B` — case-insensitive Hamming distance.

blosum62_pair_score

`blosum62_pair_score A, B` — BLOSUM62 substitution score.

kmer_count

`kmer_count SEQ, K` — total count of length-K windows.

Geographic / Map Projection

4 topics

great_circle_bearing

`great_circle_bearing LAT1, LON1, LAT2, LON2` — initial bearing in degrees.

midpoint_lat_lon

`midpoint_lat_lon LAT1, LON1, LAT2, LON2` — great-circle midpoint.

utm_zone_for

`utm_zone_for LON` — UTM zone (1..60).

area_polygon_lat_lon

`area_polygon_lat_lon POLY [, R]` — spherical-excess polygon area in m².

Fixed-Income Finance

5 topics

crr_binomial_option

`crr_binomial_option S0, K, T, R, SIGMA, N, IS_PUT` — Cox-Ross-Rubinstein price.

bond_price_clean

`bond_price_clean FACE, COUPON, N, PP_Y, YIELD` — present value.

bond_yield_to_maturity

`bond_yield_to_maturity PRICE, FACE, COUPON, N, PP_Y` — solve YTM by bisection.

modified_duration_bond

`modified_duration_bond FACE, COUPON, N, PP_Y, YIELD` — modified duration.

convexity_bond

`convexity_bond FACE, COUPON, N, PP_Y, YIELD` — bond convexity measure.

Image Quality Metrics

3 topics

ssim

`ssim A, B [, L]` — Structural Similarity Index (single window).

psnr

`psnr A, B [, MAX_VAL]` — peak SNR in dB.

mssim

`mssim A, B [, WIN, L]` — mean SSIM across non-overlapping windows.

Acoustics

4 topics

db_spl_from_pa

`db_spl_from_pa P_PA` — sound pressure level dB SPL.

a_weighting_factor

`a_weighting_factor F_HZ` — IEC 61672 A-weighting amplitude.

octave_band_center

`octave_band_center BAND` — 1 kHz · 2^band centre frequency.

semitone_ratio

`semitone_ratio` — 12-TET ratio 2^(1/12).

Population Genetics

4 topics

hardy_weinberg

`hardy_weinberg P` — (p^2, 2pq, q^2) genotype frequencies.

expected_heterozygosity

`expected_heterozygosity P_VEC` — $1 - \Sigma\ p_i^2$.

fst_simple

`fst_simple P1, P2, N1, N2` — Wright's F_ST between two populations.

allele_frequencies

`allele_frequencies GENOTYPES` — p, q from 0/1/2 genotype counts.

Epidemiology

3 topics

sir_step

`sir_step S, I, R, BETA, GAMMA, DT` — Euler step of the SIR model.

sir_r0

`sir_r0 BETA, GAMMA` — basic reproduction number β/γ.

doubling_time

`doubling_time R` — $t_2 = \ln 2 / r$.

Inequality Measures

4 topics

theil_index

`theil_index XS` — Theil T inequality index.

herfindahl_hirschman

`herfindahl_hirschman SHARES` — Σ s_i².

atkinson_index

`atkinson_index XS [, EPS]` — Atkinson inequality with parameter ε.

lorenz_curve_points

`lorenz_curve_points XS` — matrix of [cumulative-pop, cumulative-income].

APL / J Array Primitives

4 topics

iota_range

`iota_range N` — `0..N` (APL convention).

reshape_array

`reshape_array ROWS, COLS, FLAT` — APL reshape (cycles input).

grade_up

`grade_up XS` — index permutation that sorts ascending.

grade_down

`grade_down XS` — index permutation that sorts descending.

Plasma Physics

4 topics

plasma_frequency

`plasma_frequency N_E` — electron plasma angular frequency (rad/s).

debye_length

`debye_length T_K, N_E` — Debye screening length.

cyclotron_frequency

`cyclotron_frequency B [, Q, M]` — qB/m (electron defaults).

larmor_radius

`larmor_radius V_PERP, B [, Q, M]` — gyroradius (electron defaults).

String Similarity

2 topics

jaro_winkler_similarity

`jaro_winkler_similarity A, B` — JW string similarity.

metaphone_simple

`metaphone_simple S` — abridged Metaphone consonant skeleton.

Rating Systems

3 topics

elo_rating_update

`elo_rating_update R_A, R_B, SCORE_A [, K]` → [R_A′, R_B′].

glicko_rating_update

`glicko_rating_update R, RD, OPP_R, OPP_RD, SCORE` → [R′, RD′].

dice_sum_pmf

`dice_sum_pmf N, S, TARGET` — probability of N s-sided dice summing to TARGET.

Effect Sizes

3 topics

cohens_d

`cohens_d A, B` — pooled-SD standardised mean difference.

cliff_delta

`cliff_delta A, B` — non-parametric effect size in [-1, 1].

vargha_delaney_a12

`vargha_delaney_a12 A, B` — P(X > Y) + 0.5 P(X = Y).

Control Transient Response

2 topics

step_response_2nd_order

`step_response_2nd_order ZETA, OMEGA_N, T` — y(t).

overshoot_2nd_order

`overshoot_2nd_order ZETA` — peak overshoot %.

Matrix Norms

3 topics

frobenius_norm

`frobenius_norm M` — √(Σ m_ij²).

spectral_norm

`spectral_norm M` — largest singular value.

trace_matrix

`trace_matrix M` — Σ m_ii.

Network Triad Analysis

3 topics

homophily_index

`homophily_index ADJ, LABELS` — fraction of same-group edges.

dyad_census

`dyad_census ADJ` — [mutual, asymmetric, null] dyads.

triad_census

`triad_census ADJ` — [empty, edge, path, triangle] counts.

Misc Inverses

1 topics

sigmoid_inverse

`sigmoid_inverse X` — ln(x / (1-x)).

Parallel I/O

8 topics

par_lines

`par_lines PATH, { code }` — memory-map a file and scan its lines in parallel across all available CPU cores.

The file is `mmap`'d into memory rather than read sequentially, and line boundaries are detected in parallel chunks. Each line is passed to the callback as `_`. Because the file is memory-mapped, there is no read-buffer overhead and the OS kernel pages data in on demand, making `par_lines` extremely efficient for multi-gigabyte log files or CSV data. Line order within the callback is not guaranteed (lines run in parallel), so the callback should be a self-contained side-effecting operation (accumulate into a shared structure via `pchannel`, write to a file, etc.) or use `par_lines` with a reducer. For ordered processing, use `read_lines` with `pmap` instead.

```
par_lines "data.txt", fn { process }

# Count matching lines in a large log:
my $count = 0
par_lines "/var/log/syslog", fn { $count++ if /ERROR/ }
p $count

# Feed lines into a channel for downstream processing:
my ($tx, $rx) = pchannel(1000)
async { par_lines "huge.csv", fn { $tx->send(_) }
    undef $tx }
```

par_walk

`par_walk PATH, { code }` — recursively walk a directory tree in parallel, invoking the callback for every file path found.

Directory traversal is parallelized using a work-stealing thread pool: multiple directories are read concurrently, and the callback fires as each file is discovered. The path is passed as `_` (absolute). This is significantly faster than a sequential `find`-style walk on SSDs and networked filesystems where directory `readdir` latency dominates. Symlinks are not followed by default to avoid cycles. The walk visits files only — directories themselves are not passed to the callback unless you pass `dirs => 1`. Combine with `pmap` for a two-phase pattern: first collect paths with `par_walk`, then process file contents in parallel.

```
par_walk "./src", fn { p _ if /\.rs$/ }

# Collect all JSON files under a directory:
my @json_files
par_walk "/data", fn { push @json_files, _ if /\.json$/ }
p scalar @json_files

# Parallel content search:
par_walk ".", fn {
    if (/\.log$/) {
        my @hits = grep /FATAL/, rl
        p "$_: ", scalar @hits, " fatals" if @hits
    }
}
```

par_sed

`par_sed PATTERN, REPLACEMENT, @files` — perform an in-place regex substitution across multiple files in parallel.

Each file is processed by a separate worker thread: the file is read into memory, all matches of PATTERN are replaced with REPLACEMENT, and the result is written back atomically (via a temp file + rename, so readers never see a partially-written file). This is the stryke equivalent of `sed -i` but parallelized across the file list — ideal for codebase-wide refactors, log scrubbing, or bulk config updates. The pattern uses stryke regex syntax (PCRE-style). Returns the total number of substitutions made across all files.

```
par_sed qr/oldFunc/, "newFunc", glob("src/*.pl")

# Case-insensitive replace across a project:
par_sed qr/TODO/i, "DONE", par_walk(".", fn { _ if /\.rs$/ })

# Scrub sensitive data from logs:
my $n = par_sed qr/\b\d{3}-\d{2}-\d{4}\b/, "XXX-XX-XXXX", @log_files
p "redacted $n occurrences"
```

par_find_files

Recursively search a directory tree in parallel for files matching a glob pattern. Unlike `glob_par` which takes a single pattern string, `par_find_files` separates the root directory from the pattern, making it convenient when the search root is a variable. Returns a list of absolute paths to matching files.

```
my @src = par_find_files("src", "*.rs")
p scalar @src                    # count of Rust files under src/
my @tests = par_find_files(".", "*_test.pl")
@tests |> e p
my @configs = par_find_files("/etc", "*.conf")
```

par_line_count

Count lines across multiple files in parallel, returning the total line count. Each file is read and counted by a separate thread, making this dramatically faster than sequential `wc -l` for large file sets. Useful for codebase metrics, log analysis, or validating data pipeline output.

```
my @files = glob("src/**/*.rs")
my $total = par_line_count(@files)
p "$total lines of Rust"
my $logs = par_line_count(glob("/var/log/*.log"))
p "$logs log lines"
```

par_csv_read

`par_csv_read @files` — read multiple CSV files in parallel, returning an array of parsed datasets (one per file).

Each file is read and parsed by a separate worker thread using a fast Rust CSV parser that handles quoting, escaping, and UTF-8 correctly. Headers are auto-detected from the first row of each file, and each row is returned as a hashref keyed by header names. This is dramatically faster than sequential CSV parsing when you have many files — common in data engineering pipelines where data arrives as daily/hourly CSV partitions. For a single large CSV file, prefer `par_lines` with manual splitting, since `par_csv_read` parallelizes across files, not within a single file.

```
my @datasets = par_csv_read glob("data/*.csv")
for my $ds (@datasets) {
    p scalar @$ds, " rows"
}

# Merge all CSVs into one list:
my @all_rows = par_csv_read(@files) |> flat
p $all_rows[0]->{name}               # access by header

# Filter and aggregate:
my @sales = par_csv_read(glob("sales_*.csv")) |> flat
  |> grep { _->{region} eq "US" }
```

glob_par

Parallel recursive file-system glob using rayon across multiple patterns. Backed by the same zshrs glob engine as `glob`, so every zsh qualifier (`(/)`, `(.)`, `(L+1024)`, `(om[1])`, ...) is supported here too. Each pattern expands single-threaded but the patterns run concurrently — ideal for many independent searches. For full qualifier reference see `glob`.

```
my @logs = glob_par("**/*.log")
p scalar @logs              # count of log files
"**/*.rs" |> glob_par |> e p  # print all Rust files
my @imgs = glob_par("assets/**/*.{png,jpg,webp}")
my @big_recent = glob_par("**(.L+1m om[1,5])")  # 5 newest >1MB files
```

pwatch

`pwatch(PATTERN, fn { ... })` — watch a path or glob pattern for filesystem changes and invoke the callback on each event.

The watcher runs on a background thread using OS-native notifications (FSEvents on macOS, inotify on Linux) so it consumes near-zero CPU while idle. The callback receives the event path in _. The watcher continues until the returned handle is dropped or the program exits. Useful for live-reload dev servers, file-triggered pipelines, or audit logs. Combine with `debounce` or a `pchannel` if the callback is expensive and rapid bursts of events need to be coalesced.

The initial expansion goes through the full zshrs glob engine, so zsh qualifiers (`(.)`, `(/)`, `(L+1m)`, ...) work for picking what to watch. The runtime event matcher uses the qualifier-stripped pattern shape — stat-based qualifiers can't apply to a single fired event, but they did apply to the watch-set selection upfront.

```
my $w = pwatch "./src/**/*.rs", fn {
    p "changed: $_"
    rebuild()
}

my $logs = pwatch "logs/*(.)", fn { p "log updated: $_" }
my $cfg = pwatch "./config", fn { reload_config() }
sleep                               # block forever
```

File I/O

35 topics

open

Open a filehandle for reading, writing, appending, or piping. The three-argument form `open my $fh, MODE, EXPR` is strongly preferred for safety — it prevents shell injection and makes the mode explicit. Modes include `<` (read), `>` (write/truncate), `>>` (append), `+<` (read-write), `|-` (pipe to command), and `-|` (pipe from command). In stryke, always use lexical filehandles (`my $fh`) rather than bareword globals. PerlIO layers can be specified in the mode: `<:utf8`, `<:raw`, `<:encoding(UTF-16)`. Always check the return value — `open ... or die "...: $!"` is idiomatic. The `$!` variable contains the OS error message on failure. Forgetting to check `open` is one of the most common bugs in Perl code.

```
open my $fh, '<', 'data.txt' or die "open: $!"
my @lines = <$fh>
close $fh

open my $out, '>>', 'log.txt' or die "append: $!"
print $out tm "event happened\n"
close $out

open my $pipe, '-|', 'ls', '-la' or die "pipe: $!"
while (<$pipe>) { p tm _ }
```

close

Close a filehandle, flushing any buffered output and releasing the underlying OS file descriptor. Returns true on success, false on failure — and failure is more common than you might think. For write handles, `close` is where buffered data actually hits disk, so a full disk or network error may only surface at `close` time. Always check the return value when writing: `close $fh or die "close: $!"`. For pipe handles opened with `|-` or `-|`, `close` waits for the child process to exit and sets `$?` to the child's exit status. In stryke, lexical filehandles are automatically closed when they go out of scope, but explicit `close` is clearer and lets you handle errors.

```
open my $fh, '>', 'out.txt' or die $!
print $fh "done\n"
close $fh or die "write failed: $!"

open my $p, '|-', 'gzip', '-c' or die $!
print $p $data
close $p
p "gzip exited: $?" if $?
```

read

Read a specified number of bytes from a filehandle into a scalar buffer. The signature is `read FH, SCALAR, LENGTH [, OFFSET]`. Returns the number of bytes actually read (which may be less than requested at EOF or on partial reads), 0 at EOF, or `undef` on error. The optional OFFSET argument lets you append to an existing buffer at a given position, which is useful for accumulating data in a loop. For text files, the bytes are decoded according to the handle's PerlIO layer — use `<:raw` for binary data to avoid encoding transforms. In stryke, `read` works identically to Perl 5. For line-oriented input, prefer `<$fh>` or `readline` instead.

```
open my $fh, '<:raw', $path or die $!
my $buf = ''
while (read $fh, my $chunk, 4096) {
    $buf .= $chunk
}
p "total: " . length($buf) . " bytes"
close $fh
```

readline

Read one line (or all remaining lines in list context) from a filehandle. The angle-bracket operator `<$fh>` is syntactic sugar for `readline($fh)`. In scalar context, returns the next line including the trailing newline (or `undef` at EOF). In list context, returns all remaining lines as a list. The line ending is determined by `$/` (input record separator, default `\n`). Set `$/ = undef` to slurp the entire file in one read. In stryke, `readline` integrates with the pipe operator — you can pipe filehandle lines through `maps`, `greps`, and other streaming combinators. Always `chomp` after reading if you don't want trailing newlines.

```
while (my $line = <$fh>) {
    chomp $line
    p $line
}

# Slurp entire file
local $/
my $content = <$fh>
p length $content
```

eof

Test whether a filehandle has reached end-of-file. Returns 1 if the next read on the handle would return EOF, 0 otherwise. Called without arguments, `eof()` (with parens) checks the last file in the `<>` / `ARGV` stream. Called with no parens as `eof`, it tests whether the current ARGV file is exhausted but more files may follow. In stryke, `eof` is typically used in `until` loops or as a guard before `read` calls. Note that `eof` may trigger a blocking read on interactive handles (like STDIN from a terminal) to determine if data is available, so avoid calling it speculatively on interactive input. For most line-processing, `while (<$fh>)` is simpler and implicitly handles EOF.

```
until (eof $fh) {
    my $line = <$fh>
    p tm $line
}

# Process multiple files via ARGV
while (<>) {
    p "new file: $ARGV" if eof()
}
```

seek

Reposition a filehandle to an arbitrary byte offset. The signature is `seek FH, POSITION, WHENCE` where WHENCE is 0 (absolute from start), 1 (relative to current position), or 2 (relative to end of file). Use the `Fcntl` constants `SEEK_SET`, `SEEK_CUR`, `SEEK_END` for clarity. Returns 1 on success, 0 on failure. `seek` is essential for random-access file I/O — re-reading headers, skipping to known offsets in binary formats, or rewinding a file for a second pass. In stryke, `seek` flushes the PerlIO buffer before repositioning. Do not mix `seek`/`tell` with `sysread`/`syswrite` — they use separate buffering layers.

```
seek $fh, 0, 0         # rewind to start
my $header = <$fh>

seek $fh, -100, 2      # last 100 bytes
read $fh, my $tail, 100
p $tail
```

tell

Return the current byte offset of a filehandle's read/write position. Returns a non-negative integer on success, or -1 if the handle is invalid or not seekable (e.g., pipes, sockets). Useful for bookmarking a position before a speculative read so you can `seek` back if the data doesn't match expectations. In stryke, `tell` reflects the PerlIO buffered position, not the raw OS file descriptor position — so it correctly accounts for encoding layers and buffered reads. Pair with `seek` for random-access patterns.

```
my $pos = tell $fh
p "at byte $pos"

# Bookmark and restore
my $mark = tell $fh
my $line = <$fh>
unless ($line =~ /^HEADER/) {
    seek $fh, $mark, 0   # rewind to before the read
}
```

print

Write operands to the selected output handle (default `STDOUT`) without appending a newline. The output field separator `$,` is inserted between arguments, and `$\` is appended at the end — both default to empty string. You can direct output to a specific handle by passing it as the first argument with no comma: `print STDERR "msg"`. In stryke, `print` behaves identically to Perl 5 and is useful when you need precise control over output formatting, such as building progress bars, writing binary data, or emitting partial lines. For most line-oriented output, prefer `p` (say) instead since it handles the newline automatically.

```
print "no newline"
print STDERR "error msg\n"
print $fh "data to filehandle\n"
for my $pct (0:100) {
    printf "\rprogress: %3d%%", $pct
}
print "\n"
```

say

Print operands followed by an automatic newline to `STDOUT`. In stryke, `say` is always available without `-E` or `use feature 'say'` — it is a first-class builtin. The shorthand `p` is an alias for `say` and is preferred in most stryke code. When given a list, `say` joins elements with `$,` (output field separator, empty by default) and appends `$\` plus a newline. For streaming output over pipelines, combine with `e` (each) to print one element per line. Gotcha: `say` always adds a newline — if you need raw output without one, use `print` instead.

```
p "hello world"
my @names = ("alice", "bob", "eve")
@names |> e p
1:5 |> maps { _ * 2 } |> e p
```

printf

Formatted print to a filehandle (default `STDOUT`), using C-style format specifiers. The first argument is the format string with `%s` (string), `%d` (integer), `%f` (float), `%x` (hex), `%o` (octal), `%e` (scientific), `%g` (general float), and `%%` (literal percent). Width and precision modifiers work as in C: `%-10s` left-aligns in a 10-char field, `%05d` zero-pads to 5 digits, `%.2f` gives 2 decimal places. In stryke, `printf` supports all standard Perl 5 format specifiers including `%v` (version strings) and `%n` is disabled for safety. Direct output to a handle by placing it before the format: `printf STDERR "...", @args`. Unlike `sprintf`, `printf` outputs directly and returns a boolean indicating success.

```
printf "%-10s %5d\n", $name, $score
printf STDERR "error %d: %s\n", $code, $msg
printf "%08.2f\n", 3.14159   # 00003.14
printf "%s has %d item%s\n", $user, $n, $n == 1 ? "" : "s"
```

sprintf

Return a formatted string without printing it, using the same C-style format specifiers as `printf`. This is the go-to function for building formatted strings for later use — constructing log messages, building padded table columns, converting numbers to hex or binary representations, or assembling strings that will be passed to other functions. In stryke, `sprintf` is often combined with the pipe operator: `$val |> t { sprintf "0x%04x", _ } |> p`. The return value is always a string. All format specifiers from `printf` apply here.

```
my $hex = sprintf "0x%04x", 255
my $padded = sprintf "%08d", $id
my $msg = sprintf "%-20s: %s", $key, $value
my @rows = map { sprintf "%3d. %s", _, $names[_] } 0..$ #names
@rows |> e p
```

slurp

Read an entire file into memory as a single UTF-8 string, or — when the path is a glob pattern — concatenate the contents of every matching regular file. Aliases: `sl`, `cat`, `c`. Dies if the file is missing or unreadable. For binary data use `read_bytes`/`slurp_raw`.

When the path contains glob metacharacters (`* ? [ **`) or a trailing zsh qualifier suffix `(...)`, it is expanded by the zshrs glob engine and the results are concatenated. `slurp` hard-fails if the qualifier yields any non-regular file — `slurp "**(/)"` errors `slurp: not a regular file: ./sub` because slurping a directory is meaningless. To list directory paths, use `glob "**(/)"`. See `glob` for the full qualifier reference.

```
my $text = slurp("config.yaml")
my $all = c("**(.)")               # every regular file recursively, concatenated
my $logs = c("logs/*.log")          # all logs in one go
my $json = sl("data.json")
my $data = decode_json($json)
"README.md" |> sl |> length |> p    # character count
```

slurp_raw

Read an entire file into memory as raw bytes without any encoding interpretation. Unlike `slurp`, which returns a decoded UTF-8 string, `read_bytes` preserves the exact byte content — useful for binary files like images, compressed archives, or protocol buffers. The alias `slurp_raw` emphasizes the raw nature.

```
my $png = read_bytes("logo.png")
p length($png)                      # byte count
my $gz = slurp_raw("data.gz")
my $text = gunzip($gz)
p $text
```

input

Slurp all of stdin (or a filehandle) as one string.

```
my $all = input          # slurp stdin
my $fh_data = input($fh) # slurp filehandle
```

read_lines

Read a file and return its contents as a list of lines with trailing newlines stripped. This is the idiomatic way to slurp a file line-by-line in stryke without manually opening a filehandle. The short alias `rl` keeps one-liners concise. If the file does not exist, the program dies with an error message.

```
my @lines = rl("data.txt")
p scalar @lines               # line count
@lines |> grep /ERROR/ |> e p # print error lines
my $first = (rl "config.ini")[0]$1

Note: returns an empty list for an empty file.
```

append_file

Append a string to the end of a file, creating it if it does not exist. This is the safe way to add content without overwriting — useful for log files, CSV accumulation, or incremental output. The short alias `af` is convenient in pipelines. The file is opened, written, and closed atomically per call.

```
af("log.txt", "started at " . datetime_utc() . "\n")
1:5 |> e { af("nums.txt", "$_\n") }
my @data = ("a","b","c")
@data |> e { af "out.txt", "$_\n" }
```

to_file

Write a string to a file, truncating any existing content. Unlike `append_file`, this replaces the file entirely. Returns the written content so it can be used in a pipeline — write to disk and continue processing in one expression. Creates the file if it does not exist.

```
my $csv = "name,age\nAlice,30\nBob,25"
$csv |> to_file("people.csv") |> p
to_file("empty.txt", "")  # truncate a file
```

Note: the return-value-for-piping behavior distinguishes this from a plain write.

write

Output a formatted record to a filehandle using a `format` declaration. `write` looks up the format associated with the current (or specified) filehandle, evaluates the format's picture lines against the current variables, and outputs the result. This is Perl's original report-generation mechanism, predating modules like `Text::Table` and template engines. In stryke, `write` and `format` are supported for backward compatibility but are rarely used in new code — `printf`/`sprintf` or template strings are more flexible. The format name defaults to the filehandle name (e.g., format `STDOUT` is used by `write STDOUT`).

```
format STDOUT =
@<<<<<<<<<< @>>>>>>
$name,       $score
.

my ($name, $score) = ("alice", 42)
write   # outputs: alice           42
```

write_file

Write a string to a file, creating it if it does not exist or truncating it if it does. This is the complement of `slurp` — together they form a read/write pair for whole-file operations. The short alias `wf` is convenient for one-liners. The file is opened, written, and closed in a single call.

```
spurt("hello.txt", "Hello, world!\n")
wf("nums.txt", join("\n", 1:10))
"generated content" |> wf("out.txt")
my $data = slurp("in.txt")
wf("copy.txt", $data)
```

write_json

Serialize a stryke data structure (hash ref or array ref) as pretty-printed JSON and write it to a file. Creates or overwrites the target file. The short alias `wj` pairs with `rj` for round-trip JSON workflows. Useful for persisting configuration, caching API responses, or generating fixture data.

```
my %data = (name => "Alice", scores => [98, 87, 95])
wj("out.json", \%data)
my $back = rj("out.json")
p $back->{name}   # Alice
```

read_json

Read a JSON file from disk and parse it into a stryke data structure (hash ref or array ref). The short alias `rj` keeps JSON-config one-liners terse. Dies if the file does not exist or contains malformed JSON. This is the complement of `write_json`/`wj`.

```
my $cfg = rj("config.json")
p $cfg->{database}{host}
my @items = @{ rj("list.json") }
@items |> e { p _->{name} }$1

Note: numeric strings remain strings; use `+0` to coerce if needed.
```

tempfile

Create a temporary file in the system temp directory and return its absolute path as a string. The file is created with a unique name and exists on disk immediately. Use `tf` as a short alias for quick scratch files in one-liners. The caller is responsible for cleanup, though OS temp-directory reaping will eventually reclaim it.

```
my $tmp = tf()
to_file($tmp, "scratch data\n")
p rl($tmp)           # scratch data
my @all = map { tf() } 1:3  # three temp files
```

tempdir

Create a temporary directory in the system temp directory and return its absolute path. The directory is created with a unique name and is ready for use immediately. The short alias `tdr` mirrors `tf` for files. Useful for isolating multi-file operations like test fixtures, build artifacts, or staged output.

```
my $dir = tdr()
to_file("$dir/a.txt", "hello")
to_file("$dir/b.txt", "world")
my @files = glob("$dir/*.txt")
p scalar @files   # 2
```

binmode

Set the I/O layer on a filehandle, controlling how bytes are translated during reads and writes. Without a layer argument, `binmode $fh` switches the handle to raw binary mode (no CRLF translation on Windows, no encoding transforms). With a layer, `binmode $fh, ':utf8'` enables UTF-8 decoding, `binmode $fh, ':raw'` strips all layers for pure byte I/O, and `binmode $fh, ':encoding(ISO-8859-1)'` sets a specific encoding. In stryke, encoding layers can also be specified directly in `open`: `open my $fh, '<:utf8', $path`. Call `binmode` before any I/O on the handle — calling it mid-stream may produce garbled output. For binary file processing (images, archives, network protocols), always use `:raw`.

```
open my $fh, '<', $path or die $!
binmode $fh, ':utf8'
my @lines = <$fh>

open my $bin, '<', $img_path or die $!
binmode $bin, ':raw'
read $bin, my $header, 8
p sprintf "magic: %s", unpack("H*", $header)
```

fileno

Return the underlying OS file descriptor number for a filehandle, or `undef` if the handle is not connected to a real file descriptor (e.g., tied handles, in-memory handles opened to scalar refs). File descriptor numbers are small non-negative integers managed by the OS kernel: 0 is STDIN, 1 is STDOUT, 2 is STDERR. This function is mainly useful for interfacing with system calls that require raw fd numbers, checking whether two handles share the same underlying fd, or passing descriptors to child processes. In stryke, `fileno` is rarely needed in everyday code but is essential for low-level I/O multiplexing and process management.

```
my $fd = fileno STDOUT
p "stdout fd: $fd"   # 1

if (defined fileno $fh) {
    p "handle is backed by fd " . fileno($fh)
} else {
    p "not a real file descriptor"
}
```

flock

Advisory file locking for coordinating access between processes. `flock $fh, OPERATION` where OPERATION is `LOCK_SH` (shared/read lock), `LOCK_EX` (exclusive/write lock), or `LOCK_UN` (unlock). Add `LOCK_NB` for non-blocking mode: `LOCK_EX | LOCK_NB` returns false immediately if the lock is held rather than waiting. Import constants from `Fcntl`. Advisory locks are cooperative — they only work if all processes accessing the file use `flock`. In stryke, `flock` is the standard mechanism for safe concurrent file access in multi-process scripts, cron jobs, and daemons. Always unlock explicitly or let the handle close (which releases the lock). Gotcha: `flock` does not work on NFS on many systems.

```
use Fcntl ':flock'
open my $fh, '>>', 'shared.log' or die $!
flock $fh, LOCK_EX or die "lock: $!"
print $fh tm "pid $$ wrote this\n"
flock $fh, LOCK_UN
close $fh

unless (flock $fh, LOCK_EX | LOCK_NB) {
    p "file is locked by another process"
}
```

getc

Read a single character from a filehandle (default `STDIN`). Returns `undef` at EOF. The character returned respects the handle's encoding layer — under `:utf8`, `getc` returns a full Unicode character which may be multiple bytes on disk. This function is useful for interactive single-key input, parsing binary formats one byte at a time, or implementing character-level tokenizers. In stryke, `getc` blocks until a character is available on the handle. For terminal input, note that most terminals are line-buffered by default, so `getc STDIN` won't return until the user presses Enter unless you put the terminal into raw mode.

```
my $ch = getc STDIN
p "you pressed: $ch"

open my $fh, '<:utf8', $path or die $!
while (defined(my $c = getc $fh)) {
    p "char: $c (ord " . ord($c) . ")"
}
```

select

In its one-argument form, `select HANDLE` sets the default output handle for `print`, `say`, and `write`, returning the previously selected handle. This is useful for temporarily redirecting output — for example, sending diagnostics to STDERR while a report goes to a file. In its four-argument form, `select RBITS, WBITS, EBITS, TIMEOUT` performs POSIX `select(2)` I/O multiplexing, waiting for one or more file descriptors to become ready for reading, writing, or to report exceptions. The four-argument form is low-level and rarely used directly in stryke — prefer `IO::Select` or async I/O patterns for multiplexing. `select` with `$|` is also the classic way to enable autoflush on a handle.

```
my $old = select STDERR
p "this goes to stderr"
select $old

# Enable autoflush on STDOUT
select STDOUT; $| = 1
print "immediately flushed"
```

truncate

Truncate a file to a specified byte length. Accepts either a filehandle or a filename as the first argument, and the desired length as the second. `truncate $fh, 0` empties the file entirely — a common pattern when rewriting a file in place. `truncate $fh, $len` discards everything beyond byte `$len`. Returns true on success, false on failure (check `$!` for the error). In stryke, truncate works on any seekable filehandle. When using truncate to rewrite a file, remember to also `seek $fh, 0, 0` to rewind the write position — truncating does not move the file pointer. Gotcha: truncating a file opened read-only will fail.

```
open my $fh, '+<', 'data.txt' or die $!
truncate $fh, 0       # empty the file
seek $fh, 0, 0        # rewind to start
print $fh "fresh content\n"
close $fh
```

sysopen

Low-level open using POSIX flags for precise control over how a file is opened. The signature is `sysopen FH, FILENAME, FLAGS [, PERMS]`. Flags are bitwise-OR combinations from `Fcntl`: `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_CREAT`, `O_EXCL`, `O_TRUNC`, `O_APPEND`, `O_NONBLOCK`, and others. The optional PERMS argument (e.g., `0644`) sets the file mode when `O_CREAT` creates a new file, subject to the process umask. `sysopen` is the right tool when you need `O_EXCL` for atomic file creation (lock files, temp files), `O_NONBLOCK` for non-blocking I/O, or other flags that `open` cannot express. In stryke, prefer three-argument `open` for routine file access and reserve `sysopen` for cases requiring specific POSIX semantics.

```
use Fcntl
# Atomic create — fails if file already exists
sysopen my $lock, '/tmp/my.lock', O_WRONLY|O_CREAT|O_EXCL, 0644
    or die "already running: $!"
print $lock $$

sysopen my $log, 'app.log', O_WRONLY|O_APPEND|O_CREAT, 0644
    or die "open log: $!"
```

sysread

Low-level unbuffered read directly from a file descriptor, bypassing PerlIO buffering layers. The signature is `sysread FH, SCALAR, LENGTH [, OFFSET]`. Returns the number of bytes read, 0 at EOF, or `undef` on error. Unlike `read`, `sysread` issues a single `read(2)` system call and may return fewer bytes than requested (short read). This is the right choice for sockets, pipes, and non-blocking I/O where you need precise control over how many system calls occur and cannot tolerate buffering. In stryke, never mix `sysread`/`syswrite` with buffered I/O (`print`, `read`, `<$fh>`) on the same handle — the buffered and unbuffered positions will diverge and produce corrupted reads.

```
open my $fh, '<:raw', $path or die $!
my $buf = ''
while (my $n = sysread $fh, $buf, 4096, length($buf)) {
    p "read $n bytes (total: " . length($buf) . ")"
}
close $fh
```

syswrite

Low-level unbuffered write directly to a file descriptor, bypassing PerlIO buffering layers. The signature is `syswrite FH, SCALAR [, LENGTH [, OFFSET]]`. Returns the number of bytes actually written, which may be less than requested on non-blocking handles or when writing to pipes/sockets (short write). Returns `undef` on error. Like `sysread`, this issues a single `write(2)` system call and must not be mixed with buffered I/O on the same handle. In stryke, `syswrite` is essential for socket programming, IPC, and performance-critical binary output where you need to avoid double-buffering. Always check the return value and handle short writes in a loop for robust code.

```
my $data = "hello, world"
my $n = syswrite $fh, $data
p "wrote $n bytes"

# Robust write loop for sockets
my $off = 0
while ($off < length $data) {
    my $written = syswrite $fh, $data, length($data) - $off, $off
    die "syswrite: $!" unless defined $written
    $off += $written
}
```

sysseek

Low-level seek on a file descriptor, bypassing PerlIO buffering. The signature is `sysseek FH, POSITION, WHENCE` with the same WHENCE values as `seek` (0=start, 1=current, 2=end). Returns the new position as a true value, or `undef` on failure. Unlike `seek`, `sysseek` does not flush PerlIO buffers — it operates directly on the underlying OS file descriptor. Use `sysseek` when working with `sysread`/`syswrite` for consistent positioning. In stryke, `sysseek` with `SEEK_CUR` and position 0 is an idiom for querying the current fd position without moving: `my $pos = sysseek $fh, 0, 1`.

```
sysseek $fh, 0, 0   # rewind to start
sysread $fh, my $buf, 512

my $pos = sysseek $fh, 0, 1
p "fd position: $pos"
```

format

Declare a picture-format template for generating fixed-width text reports. The syntax is `format NAME = ... .` where each line alternates between picture lines (containing field placeholders) and argument lines (listing the variables to fill in). Placeholders include `@<<<<` (left-align), `@>>>>` (right-align), `@||||` (center), `@###.##` (numeric with decimal), and `@*` (multiline block fill). The format ends with a lone `.` on its own line. In stryke, formats are a legacy feature preserved for compatibility with Perl 5 code — for new reports, prefer `printf`/`sprintf` for simple alignment or a templating module for complex layouts. Formats interact with the special variables `$~` (current format name), `$^` (header format), and `$=` (lines per page).

```
format REPORT =
@<<<<<<<<<<<<<<< @>>>>> @ ###.##
$item,            $qty,  $price
.

my ($item, $qty, $price) = ("Widget", 100, 9.99)
my $old = select REPORT
$~ = 'REPORT'
write REPORT
select $old
```

formline

Format a line of output according to a picture template, appending the result to the `$^A` (format accumulator) variable. This is the low-level engine behind Perl's `format`/`write` report-generation system. Template characters like `@<<<` (left-justify), `@>>>` (right-justify), and `@###.##` (numeric) control field placement.

```
$^A = ""
formline("@<<<< @>>>>>\n", "Name", "Score")
formline("@<<<< @>>>>>\n", "Alice", 98)
formline("@<<<< @>>>>>\n", "Bob", 85)
p $^A
```

Strings

26 topics

chomp

`chomp STRING` — remove the trailing record separator (usually `\n`) from a string in place and return the number of characters removed.

`chomp` is the idiomatic way to strip newlines after reading input; unlike `chop`, it only removes the value of `$/` (the input record separator), so it is safe to call on strings that do not end with a newline — it simply does nothing. You can also `chomp` an entire array to strip every element at once. In stryke, `chomp` operates on UTF-8 strings and respects multi-byte `$/` values. Prefer `chomp` over `chop` in virtually all input-processing code; `chop` is only for when you truly need to remove an arbitrary trailing character regardless of what it is.

```
my $line = <STDIN>
chomp $line
p $line
chomp(my @lines = <$fh>)  # strip all at once
p scalar @lines
```

chop

`chop STRING` — remove and return the last character of a string, modifying the string in place.

Unlike `chomp`, `chop` unconditionally removes whatever the final character is — newline, letter, digit, or even a multi-byte UTF-8 codepoint in stryke. The return value is the removed character. This makes `chop` useful for peeling off known trailing delimiters or building parsers that consume input character-by-character, but dangerous for general newline removal because it will silently eat a real character if the string does not end with `\n`. When called on an array, `chop` removes the last character of every element and returns the last one removed.

```
my $s = "hello!"
my $last = chop $s  # $last = "!", $s = "hello"
p $s
my @words = ("foo\n", "bar\n")
chop @words  # strips trailing newline from each
@words |> e p
```

length

`length STRING` — return the byte length of a string. Perl 5 builtin; pairs with `index` / `rindex` / `substr` (all byte-based) so positions and lengths share a coordinate system.

`length "hello"` is 5; `length "□"` is 3 (the box-drawing char takes 3 bytes in UTF-8). For codepoint count, use `len` (stryke extension), which mirrors `[i]` / `[a:b]` slice semantics.

```
p length "hello"      # 5
p length "□"          # 3 (bytes)
p len "□"             # 1 (codepoint)
p length "\x{1F600}"  # 4 (bytes — emoji is 4-byte UTF-8)
p len "\x{1F600}"     # 1 (codepoint)
```

substr

`substr STRING, OFFSET [, LENGTH [, REPLACEMENT]]` — extract or replace a substring.

`substr` is stryke's Swiss-army knife for positional string manipulation. With two arguments it extracts from OFFSET to end; with three it extracts LENGTH characters; with four it replaces that range with REPLACEMENT and returns the original extracted

portion. Negative OFFSET counts from the end of the string (`substr $s, -3` gives the last three characters). In stryke, offsets are UTF-8 character positions, making it safe for multi-byte text. `substr` as an lvalue (`substr($s, 0, 1) = "X"`) is also supported for in-place mutation. Prefer `substr` over regex when you know exact positions — it is faster and clearer.

```
my $s = "hello world"
p substr $s, 0, 5              # hello
p substr $s, -5                # world
substr $s, 6, 5, "stryke"     # $s = "hello stryke"
p $s
```

index

`index STRING, SUBSTRING [, POSITION]` — return the zero-based byte position of the first occurrence of SUBSTRING within STRING, or -1 if not found. Pairs with `substr` and `length` (both byte-based) for Perl 5 compat.

The optional POSITION argument lets you start the search at a given offset, which is essential for scanning forward through a string in a loop (call `index` repeatedly, advancing POSITION past each hit). For multi-byte strings where you want codepoint positions paired with `[i]` / `[a:b]` slice operators, use `cindex` (stryke extension) instead — `index` is byte-indexed because `length` and `substr` are, and changing it would break Perl 5 binary-protocol code.

```
my $i = index "hello world", "world"  # 6
p $i
my $s = "a.b.c.d"
my $pos = 0
while (($pos = index $s, ".", $pos) != -1) {
    p "dot at $pos"
    $pos++
}
```

rindex

`rindex STRING, SUBSTRING [, POSITION]` — return the zero-based byte position of the last occurrence of SUBSTRING within STRING, searching backward from POSITION (or the end). Mirror of `index`; pairs with `substr` and `length` for Perl 5 compat.

`rindex` searches from right to left, making it ideal for extracting file extensions, final path components, or the last delimiter in a string. The optional POSITION argument caps how far right the search begins — characters after POSITION are ignored. Returns -1 when the substring is not found. For codepoint positions paired with `[]` slice operators, use `crindex`.

```
my $path = "foo/bar/baz.tar.gz"
my $i = rindex $path, "/"     # 7
p substr $path, $i + 1        # baz.tar.gz
my $ext = rindex $path, "."   # 15
p substr $path, $ext + 1      # gz
```

split

`split /PATTERN/, STRING [, LIMIT]` — divide STRING into a list of substrings by splitting on each occurrence of PATTERN.

`split` is one of the most-used string functions. The PATTERN is a regex, so you can split on character classes, alternations, or lookaheads. A LIMIT caps the number of returned fields; the final field contains the unsplit remainder. The special pattern `" "` (a single space string, not regex) mimics `awk`-style splitting: it strips leading whitespace and splits on runs of whitespace. In stryke, `split` integrates with `|>` pipelines, accepting piped-in strings for ergonomic one-liners. Trailing empty fields are removed by default; pass -1 as LIMIT to preserve them.

```
my @parts = split /,/, "a,b,c"
"one:two:three" |> split /:/ |> e p   # one two three
my ($user, $domain) = split /@/, $email, 2
p "$user at $domain"
```

join

`join SEPARATOR, LIST` — concatenate all elements of LIST into a single string, placing SEPARATOR between each pair of adjacent elements.

`join` is the inverse of `split`. It stringifies each element before joining, so mixing numbers and strings is fine. An empty separator (`join "", @list`) concatenates without gaps. In stryke, `join` works naturally with `|>` pipelines: a range or filtered list can be piped directly into `join` with a separator argument. This is the standard way to build CSV lines, path strings, or human-readable lists from arrays. `join` never adds a trailing separator — if you need one, append it yourself.

```
my $csv = join ",", @fields
1:5 |> join "-" |> p   # 1-2-3-4-5
my @parts = ("usr", "local", "bin")
p join "/", "", @parts   # /usr/local/bin
```

uc

`uc STRING` — return a fully uppercased copy of the string.

`uc` performs Unicode-aware uppercasing in stryke, correctly handling multi-byte characters and locale-sensitive transformations. It returns a new string without modifying the original. Use `uc` for normalizing strings before comparison, formatting headers, or transforming pipeline output. In stryke `|>` chains, combine with `maps` or `t` for streaming uppercase transforms. For uppercasing only the first character (e.g., sentence capitalization), use `ucfirst` instead.

```
p uc "hello"  # HELLO
@words |> maps { uc } |> e p
"whisper" |> t uc |> p   # WHISPER
```

lc

`lc STRING` — return a fully lowercased copy of the string.

`lc` performs Unicode-aware lowercasing in stryke, so `lc "\x{C4}"` (capital A with diaeresis) correctly returns the lowercase form. It does not modify the original string — it returns a new one. This is the standard way to normalize strings for case-insensitive comparison: `if (lc $a eq lc $b)`. In stryke `|>` pipelines, wrap `lc` with `t` to apply it as a streaming transform. For lowercasing only the first character, use `lcfirst` instead.

```
p lc "HELLO"               # hello
"SHOUT" |> t lc |> t rev |> p  # tuohs
my @norm = map lc, @words
@norm |> e p
```

ucfirst

`ucfirst STRING` — return a copy of the string with only the first character uppercased, leaving the rest unchanged.

This is the standard way to capitalize the first letter of a word for display, title-casing, or converting camelCase to PascalCase. In stryke, `ucfirst` is Unicode-aware and correctly handles multi-byte leading characters. It returns a new string and does not modify the original. For full uppercasing, use `uc`. A common idiom is `ucfirst lc $word` to normalize a word to "Title" form.

```
p ucfirst "hello"            # Hello
p ucfirst lc "hELLO"         # Hello
my @titled = map { ucfirst lc } @raw
@titled |> join " " |> p
```

lcfirst

`lcfirst STRING` — return a copy of the string with only the first character lowercased, leaving the rest unchanged.

This is useful for converting PascalCase identifiers to camelCase, or for formatting output where only the initial letter matters. In stryke, `lcfirst` is Unicode-aware, so it handles accented capitals and multi-byte first characters correctly. Like `lc`, it returns a new string rather than modifying in place. If you need the entire string lowercased, use `lc` instead.

```
p lcfirst "Hello"    # hello
p lcfirst "XMLParser" # xMLParser
my @camel = map lcfirst, @PascalNames
@camel |> e p
```

chr

`chr NUMBER` — return the character represented by the given ASCII or Unicode code point.

`chr` is the inverse of `ord`: `chr(ord($c)) eq $c` always holds. It accepts any non-negative integer and returns a single-character string. In stryke, values above 127 produce valid UTF-8 characters, so `chr 0x1F600` gives you a smiley emoji with no special encoding gymnastics. For string literals, stryke supports all standard escapes: `\x{hex}`, `\u{hex}`, `\o{oct}`, `\NNN` (octal), `\cX` (control), `\N{U+hex}`, `\N{UNICODE NAME}`, plus case modifiers `\U..\E`, `\L..\E`, `\u`, `\l`, `\Q..\E`.

```
p chr 65       # A
p chr 0x1F600  # smiley emoji
p "\u{0301}"   # combining acute accent
p "\N{SNOWMAN}" # ☃
my @abc = map { chr(_ + 64) } 1:26
@abc |> join "" |> p  # ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

ord

`ord STRING` — return the numeric (Unicode code point) value of the first character of the string.

`ord` is the inverse of `chr`: `ord(chr($n)) == $n` always holds. When passed a multi-character string, only the first character is examined — the rest are ignored. In stryke, `ord` returns the full Unicode code point, not just 0-255, so `ord "\u{1F600}"` returns 128512. This is useful for character classification, building lookup tables, or implementing custom encodings. For ASCII checks, `ord($c) >= 32 && ord($c) <= 126` tests printability.

```
p ord "A"          # 65
p ord "\n"         # 10
p ord "\u{1F600}"  # 128512
my @codes = map ord, split //, "hello"
@codes |> e p      # 104 101 108 108 111
```

hex

`hex STRING` — interpret a hexadecimal string and return its numeric value.

The leading `0x` prefix is optional: both `hex "ff"` and `hex "0xff"` return 255. The string is case-insensitive, so `hex "DeAdBeEf"` works fine. If the string contains non-hex characters, stryke warns and converts up to the first invalid character. This is the standard way to parse hex-encoded values from config files, color codes, or protocol dumps. For the reverse operation (number to hex string), use `sprintf "%x"`. Note that `hex` always returns a number, never a string — arithmetic is immediate.

```
my $n = hex "deadbeef"
printf "0x%x = %d\n", $n, $n
p hex "ff"   # 255
"cafe" |> t { hex } |> p   # 51966
```

oct

`oct STRING` — interpret an octal, hexadecimal, or binary string and return its numeric value.

`oct` is the multi-base cousin of `hex`: it auto-detects the base from the prefix. Strings starting with `0b` are binary, `0x` are hex, and bare digits or `0`-prefixed digits are octal. This makes it the go-to for parsing Unix file permissions (`oct "0755"` gives 493), binary literals, or any string where the radix is embedded in the value. In stryke, `oct` handles arbitrarily large integers via the same big-number pathway as other arithmetic. A common gotcha: `oct "8"` warns because 8 is not a valid octal digit — use `hex` or a bare numeric literal instead.

```
p oct "0755"    # 493
p oct "0b1010"  # 10
p oct "0xff"    # 255
my $perms = oct "644"
p sprintf "%04o", $perms  # 0644
```

quotemeta

`quotemeta STRING` — escape all non-alphanumeric characters with backslashes, returning a string safe for interpolation into a regex.

This is essential when building dynamic regexes from user input: without `quotemeta`, characters like `.`, `*`, `(`, `)`, `[`, and `\` would be interpreted as regex metacharacters, leading to either broken patterns or security vulnerabilities (ReDoS). The equivalent inline syntax is `\Q..\E` inside a regex. In stryke, `quotemeta` handles the full Unicode range, escaping any character that is not `[A-Za-z0-9_]`. Use it liberally whenever you splice user-provided strings into patterns.

```
my $input = "file (1).txt"
my $safe = quotemeta $input
p $safe  # file\ \(1\)\.txt
my $found = ("file (1).txt" =~ /^$safe$/)
p $found  # 1
```

trim

`trim STRING` or `trim LIST` — strip leading and trailing ASCII whitespace.

When given a single string, `trim` returns the stripped result. When given a list or used in a pipeline, it operates in streaming mode, trimming each element individually as it flows through. Whitespace includes spaces, tabs, carriage returns, and newlines. The `tm` alias is convenient in pipeline chains where brevity matters.

```
" hello " |> tm |> p                  # "hello"
@raw |> tm |> e p
slurp("data.csv") |> ln |> tm |> e p
```

lines

`lines STRING` / `ln STRING` — split a string on newline boundaries, returning a streaming iterator of lines.

Each line is yielded without the trailing newline character. When piped from `slurp`, this gives you a lazy line-by-line view of a file without loading all lines into memory at once. Both `\n` and `\r\n` line endings are handled. The `ln` alias keeps pipelines compact.

```
slurp("data.txt") |> lines |> e p
my @rows = lines $multiline_str
$body |> ln |> grep /TODO/ |> e p
```

words

`words` (alias `wd`) splits a string on whitespace boundaries and returns the resulting list of words. It handles leading, trailing, and consecutive whitespace gracefully — unlike a naive `split / /`, it never produces empty strings. This is the idiomatic way to tokenize a line of text in stryke, and the short alias `wd` keeps pipelines compact. It is equivalent to Perl's `split ' '` behavior.

```
my @w = wd "  hello   world  "
p @w       # hello, world
my $line = "  foo  bar  baz  "
my $count = scalar wd $line
p $count   # 3
rl("data.txt") |> map { scalar wd _ } |> e p
```

chars

`chars STRING` / `ch STRING` — split a string into individual characters, returning a streaming iterator.

Each Unicode grapheme cluster is yielded as a separate string element. This is useful for character-level processing such as frequency counting, transliteration, or building character n-grams. In pipeline mode the characters stream one at a time, so you can chain with `take`, `grep`, or `with_index` without materializing the full character array.

```
"hello" |> chars |> e p               # h e l l o
my @c = chars "abc"
"emoji: \x{1F600}" |> ch |> e p
```

snake_case

`snake_case` (alias `sc`) converts a string from any common casing convention — camelCase, PascalCase, kebab-case, or mixed — into snake_case, where words are lowercase and separated by underscores. This is the standard naming convention for Perl and

Rust variables and function names. Consecutive uppercase letters in acronyms are handled intelligently (e.g., `parseHTTPResponse` becomes `parse_http_response`).

```
p sc "camelCase"          # camel_case
p sc "PascalCase"         # pascal_case
p sc "kebab-case"         # kebab_case
p sc "parseHTTPResponse"  # parse_http_response
my @methods = qw(getUserName setEmailAddr)
@methods |> map sc |> e p
```

camel_case

`camel_case` (alias `cc`) converts a string from any casing convention into camelCase, where the first word is lowercase and subsequent words are capitalized with no separators. This is the standard naming convention for JavaScript variables and Java methods. The function handles underscores, hyphens, and spaces as word boundaries and strips them during conversion.

```
p cc "snake_case"         # snakeCase
p cc "kebab-case"         # kebabCase
p cc "PascalCase"         # pascalCase
p cc "hello world"        # helloWorld
my @cols = qw(first_name last_name email_addr)
@cols |> map cc |> e p
```

kebab_case

`kebab_case` (alias `kc`) converts a string from any casing convention into kebab-case, where words are lowercase and separated by hyphens. This is the standard naming convention for CSS classes, URL slugs, and CLI flag names. Like `snake_case`, it intelligently handles acronyms and mixed-case input. The function treats underscores, spaces, and case transitions as word boundaries.

```
p kc "camelCase"          # camel-case
p kc "PascalCase"         # pascal-case
p kc "snake_case"         # snake-case
p kc "parseHTTPResponse"  # parse-http-response
my $slug = kc $title      # URL-safe slug
```

study

`study STRING` — hint to the regex engine that the given string will be matched against multiple patterns, allowing it to build an internal lookup table for faster matching.

In classic Perl, `study` builds a linked list of character positions so subsequent regex matches can skip impossible starting points. In practice, modern regex engines (including stryke's Rust-based engine) already perform these optimizations internally, so `study` is effectively a no-op in stryke — calling it is harmless but provides no measurable speedup. It exists for compatibility with Perl code that uses it. You only need `study` when porting legacy scripts that call it; do not add it to new stryke code expecting a performance benefit.

```
study $text   # no-op in stryke, kept for compatibility
my @hits = grep { /$pattern/ } @lines
p scalar @hits
```

pos

`pos SCALAR` — gets or sets the position where the next `m//g` (global match) will start searching in the given string. After a successful `m//g` match, `pos` returns the offset just past the end of the match. You can assign to `pos($s)` to manually reposition the search. If the last `m//g` failed, `pos` resets to `undef`. This is essential for writing lexers or tokenizers that consume a string incrementally with `\G`-anchored patterns. In stryke, `pos` tracks per-scalar state just like Perl.

```
my $s = "abcabc"
while ($s =~ /a/g) { p pos($s) }
# 1 4
pos($s) = 0   # reset to scan again
p pos($s)     # 0
```

Arrays & Hashes

20 topics

push

`push @array, LIST` — appends one or more elements to the end of an array and returns the new length. This is the primary way to grow arrays in stryke and works identically to Perl's builtin. You can push scalars, lists, or even the result of a pipeline. In stryke, `push` is O(1) amortized thanks to the underlying Rust `Vec`.

```
my @q
push @q, 1:3
push @q, "four", "five"
p scalar @q   # 5
@q |> e p     # 1 2 3 four five
```

Returns the new element count, so `my $len = push @arr, $val;` is valid.

pop

`pop @array` — removes and returns the last element of an array, or `undef` if the array is empty. This is the complement of `push` and together they give you classic stack (LIFO) semantics. In stryke the operation is O(1) because the underlying Rust `Vec::pop` simply decrements the length. When called without an argument inside a subroutine, `pop` operates on `@_`; at file scope it operates on `@ARGV`, matching standard Perl behavior.

```
my @stk = 1:5
my $top = pop @stk
p $top          # 5
p scalar @stk   # 4
@stk |> e p     # 1 2 3 4
```

shift

`shift @array` — removes and returns the first element of an array, sliding all remaining elements down by one index. Like `pop`, it returns `undef` on an empty array. Without an explicit argument it defaults to `@_` inside subroutines and `@ARGV` at file scope. Because every element must be moved, `shift` is O(n); if you only need LIFO access, prefer `push`/`pop`. Use `shift` when processing argument lists or implementing FIFO queues.

```
my @args = @ARGV
my $cmd = shift @args
p $cmd
@args |> e p   # remaining arguments
```

unshift

`unshift @array, LIST` — prepends one or more elements to the beginning of an array and returns the new length. It is the counterpart of `push` for the front of the array. Like `shift`, this is O(n) because existing elements must be moved to make room. When you need to build an array in reverse order or maintain a priority queue with newest items first, `unshift` is the idiomatic choice. You can pass multiple values and they will appear in the same order they were given.

```
my @log = ("b", "c")
unshift @log, "a"
@log |> e p   # a b c
my $len = unshift @log, "x", "y"
p $len        # 5
@log |> e p   # x y a b c
```

splice

`splice @array, OFFSET [, LENGTH [, LIST]]` — the Swiss-army knife for array mutation. It can insert, remove, or replace elements at any position in a single call. With just an offset it removes everything from that point to the end. With offset and length it removes that many elements. With a replacement list it inserts those elements in place of the removed ones. The return value is the list of removed elements, which is useful for saving what you cut. In stryke this compiles down to Rust's `Vec::splice` and `Vec::drain`.

```
my @a = 1:5
my @removed = splice @a, 1, 2, 8, 9
@a |> e p        # 1 8 9 4 5
@removed |> e p  # 2 3
splice @a, 2     # remove from index 2 onward
@a |> e p        # 1 8
```

splice_last

`splice_last @array, OFFSET [, LENGTH [, LIST]]` — same shape as `splice`, but returns the last removed element instead of the full removed list (or `undef` if nothing was removed). The stryke spelling of Perl's `scalar splice(@a, off, n)` for `--no-interop` mode where `scalar` is rejected at parse time. The array is still mutated in place, exactly like `splice`. Aliases: `splice1`, `spl_last`. Desugars to `tail(splice(...))` so any list-context property of `splice` is preserved.

```
my @a = 10:50:10
my $last = splice_last @a, 1, 2   # removes (20, 30), returns 30
@a |> e p                          # 10 40 50
my $popped = splice_last @b, 0, 0  # removes nothing, returns undef
```

keys

`keys %hash` — returns the list of all keys in a hash in no particular order. When called on an array, it returns the list of valid indices (0 to `$#array`). In scalar context, `keys` returns the number of keys. Calling `keys` resets the `each` iterator on that hash, which is the standard way to restart iteration. In stryke, this calls Rust's `HashMap::keys()` and collects into a `Vec`. For sorted output, chain with `sort` via the pipe operator.

```
my %env = (HOME => "/root", USER => "me")
keys(%env) |> sort |> e p   # HOME USER
p scalar keys %env           # 2
my @a = (10, 20, 30)
keys(@a) |> e p              # 0 1 2
```

values

`values %hash` — returns the list of all values in a hash in no particular order (matching the order of `keys` for the same hash state). When called on an array, it returns the array elements themselves. In scalar context, returns the count of values. Like `keys`, calling `values` resets the `each` iterator. In stryke this maps to Rust's `HashMap::values()`. Combine with `sum`, `sort`, or pipeline operators for common aggregation patterns.

```
my %scores = (alice => 90, bob => 85)
p sum(values %scores)   # 175
values(%scores) |> sort { $_1 <=> $_0 } |> e p   # 90 85
```

each

`each %hash` — returns the next (key, value) pair from a hash as a two-element list, or an empty list when the iterator is exhausted. Each hash has its own internal iterator, which is reset when you call `keys` or `values` on the same hash. This is memory-efficient for large hashes because it does not build the full key list. Gotcha: do not add or delete keys while iterating with `each`; that can cause skipped or duplicated entries. In stryke, the iteration order is non-deterministic (Rust `HashMap` order).

```
my %h = (a => 1, b => 2, c => 3)
while (my ($k, $v) = each %h) { p "$k=$v" }
# output order varies: a=1 b=2 c=3
```

delete

`delete EXPR` — removes a key-value pair from a hash or an element from an array and returns the removed value (or `undef` if it did not exist). For hashes this is the only way to truly remove a key; assigning `undef` to `$h{key}` leaves the key in place. For arrays, `delete` sets the element to `undef` but does not shift indices (use `splice` if you need to close the gap). In stryke, hash deletion maps to Rust's `HashMap::remove`. You can delete multiple keys at once with a hash slice: `delete @h{@keys}`.

```
my %h = (x => 1, y => 2, z => 3)
my $old = delete $h{x}
p $old   # 1
p exists $h{x} ? "yes" : "no"   # no
delete @h{"y", "z"}   # delete multiple keys
```

exists

`exists EXPR` — tests whether a specific key is present in a hash or an index is present in an array, regardless of whether the associated value is `undef`. This is different from `defined`: a key can exist but hold `undef`. Use `exists` to check hash membership before accessing a value to avoid autovivification side effects. In stryke, `exists` on a hash compiles to Rust's `HashMap::contains_key`, and on an array it checks bounds. You can also use it with nested structures: `exists $h{a}{b}` only checks the final level.

```
my %h = (a => 1, b => undef)
p exists $h{b} ? "yes" : "no"   # yes (key present, value undef)
p exists $h{c} ? "yes" : "no"   # no
my @a = (10, 20, 30)
p exists $a[5] ? "yes" : "no"   # no
```

scalar

`scalar EXPR` — forces the expression into scalar context. The most common use is `scalar @array` to get the element count instead of the list of elements. In stryke, scalar context on an array returns its length as a Rust `usize`. You can also use `scalar` on a hash to get the number of key-value pairs, or on a function call to force it to return a single value. This is essential when you want a count inside string interpolation or as a function argument where list context would be ambiguous.

```
my @items = ("a", "b", "c")
p scalar @items   # 3
p "count: " . scalar @items   # count: 3
my %h = (x => 1, y => 2)
p scalar %h       # 2
```

defined

`defined EXPR` — returns true if the expression has a value that is not `undef`. This is the canonical way to distinguish between "no value" and "a value that happens to be false" (such as `0`, `""`, or `"0"`). Use `defined` before dereferencing optional values or checking return codes from functions that return `undef` on failure. In stryke, `defined` compiles to a Rust `Option::is_some()` check internally. Note that `defined` on an aggregate (hash or array) is deprecated in Perl 5 and is a no-op in stryke.

```
my $x = undef
p defined($x) ? "yes" : "no"   # no
$x = 0
p defined($x) ? "yes" : "no"   # yes
my $val = fn_that_may_fail()
p $val if defined $val
```

undef

`undef` — the undefined value, representing the absence of a value. As a function, `undef $var` explicitly undefines a variable, freeing any value it held. `undef` is falsy in boolean context and triggers "use of uninitialized value" warnings under `use warnings`. In stryke, `undef` maps to Rust's `None` in an `Option` type internally. Use `undef` to reset variables, signal missing return values, or clear hash entries without deleting the key.

```
my $x = 42
undef $x
p defined($x) ? "def" : "undef"   # undef
fn maybe { return undef if !@_
    return $_[0] }
p defined(maybe()) ? "got" : "nothing"   # nothing
```

ref

`ref EXPR` — returns a string indicating the reference type of the value, or an empty string if it is not a reference. Common return values are `SCALAR`, `ARRAY`, `HASH`, `CODE`, `REF`, and `Regexp`. For blessed objects it returns the class name. Use `ref` to dispatch on data type or validate arguments in polymorphic functions. In stryke, `ref` inspects the Rust enum variant of the underlying value. Note that `ref` does not recurse; it only tells you the top-level type.

```
my $r = [1, 2, 3]
p ref($r)       # ARRAY
p ref(\%ENV)    # HASH
p ref(\&main)   # CODE
p ref(42)       # (empty string)
```

bless

Associate a reference with a package name, turning it into an object of that class. The blessed reference can then have methods called on it via `->`. The second argument defaults to the current package. This is the foundation of Perl's object system.

```
fn new($class, %args) {
    bless { %args }, $class
}
my $obj = new("Dog", name => "Rex", breed => "Lab")
p ref($obj)          # Dog
p $obj->{name}       # Rex
```

tie

Bind a variable to an implementing class so that all accesses (read, write, delete, etc.) are intercepted by methods on that class. This is Perl's mechanism for transparent object-backed variables — tied hashes can be backed by a database, tied scalars can validate on assignment, etc. Use `untie` to remove the binding.

```
tie my %db, 'DB_File', 'cache.db'
$db{key} = "value"         # writes to disk
p $db{key}                  # reads from disk
untie %db
```

prototype

Return the prototype string of a named function, or `undef` if the function has no prototype. Prototypes control how arguments are parsed at compile time — they influence context and reference-passing behavior. Useful for introspection and metaprogramming.

```
p prototype("CORE::push")     # \@@
p prototype("CORE::map")      # &@
fn greet($name) { p "hi $name" }
p prototype(\&greet)          # undef (signatures, no proto)
```

wantarray

`wantarray()` — returns true if the current subroutine was called in list context, false in scalar context, and `undef` in void context. This lets a function adapt its return value to the caller's expectations. A common pattern is returning a list in list context and a count or reference in scalar context. In stryke, use `fn` to define subroutines and `wantarray()` inside them just like in Perl. Note that `wantarray` only reflects the immediate call site, not nested contexts.

```
fn ctx { wantarray() ? "list" : "scalar" }
my @r = ctx()
p $r[0]   # list
my $r = ctx()
p $r      # scalar
fn flexible { wantarray() ? (1, 2, 3) : 3 }
my @all = flexible()
p scalar @all   # 3
my $cnt = flexible()
p $cnt          # 3
```

caller

`caller [LEVEL]` — returns information about the calling subroutine's context. In list context it returns `(package, filename, line)` for the given call-stack level (default 0, the immediate caller). With higher levels you can walk up the call stack for debugging or generating stack traces. In scalar context, `caller` returns just the package name. In stryke, `caller` works with `fn`-defined subroutines and integrates with the runtime's frame tracking. This is invaluable for building custom error reporters or trace utilities.

```
fn here {
    my ($pkg, $f, $ln) = caller()
    p "$f:$ln"
}
here()   # prints current file:line
fn deep {
    my ($pkg, $f, $ln) = caller(1)
    p "grandparent: $f:$ln"
}
```

Control Flow

17 topics

if

The fundamental conditional construct. Evaluates its condition in boolean context and executes the block if true. stryke supports both the block form `if (COND) { BODY }` and the postfix form `EXPR if COND`, which is idiomatic for single-statement guards. The condition can be any expression — numbers (0 is false), strings (empty and `"0"` are false), undef (false), and references (always true). Postfix `if` cannot have `elsif`/`else` — use the block form for multi-branch logic.

```
my $n = 42
if ($n > 0) { p "positive" }
p "big" if $n > 10          # postfix — clean one-liner
my $label = "even" if $n % 2 == 0
p "got: $n" if defined $n   # guard against undef
```

elsif

Chain additional conditions after an `if` block without nesting. Each `elsif` is tested in order; the first one whose condition is true has its block executed, and the rest are skipped. There is no limit on the number of `elsif` branches. In stryke, prefer `match` for complex multi-branch dispatch since it supports pattern matching, destructuring, and guards — but `elsif` remains the right tool for simple linear condition chains. Note: it is `elsif`, not `elseif` or `else if` — the latter is a syntax error.

```
fn classify($n) {
    if    ($n < 0)   { "negative" }
    elsif ($n == 0)  { "zero" }
    elsif ($n < 10)  { "small" }
    elsif ($n < 100) { "medium" }
    else             { "large" }
}
1:200 |> map classify |> frequencies |> dd
```

else

The final fallback branch of an `if`/`elsif` chain, executed when no preceding condition was true. Every `if` can have at most one `else`, and it must come last. For ternary-style expressions, use `COND ? A : B` instead of an `if`/`else` block — it composes better in `|>` pipes and assignments.

```
if ($n % 2 == 0) { p "even" }
else             { p "odd" }

# Ternary is often cleaner in pipes:
1:10 |> map { _ % 2 == 0 ? "even" : "odd" } |> e p
```

unless

A negated conditional — executes the block when the condition is *false*. This reads more naturally than `if (!COND)` for guard clauses and early returns. stryke supports both block and postfix forms. Convention: use `unless` for simple negative guards; avoid `unless` with complex compound conditions, as double-negatives hurt readability. There is no `unlessif` — use `if`/`elsif` chains for multi-branch logic.

```
unless ($ENV{QUIET}) { p "verbose output" }
p "missing!" unless -e $path   # postfix guard
die "no input" unless @ARGV
return unless defined $val    # early return pattern
```

for

Iterate over a list, binding each element to a loop variable (or _ by default). `for` and `foreach` are interchangeable keywords. The loop variable is automatically localized and aliases the original element — modifications to _ inside the loop mutate the list in-place. In stryke, `for` loops work with ranges, arrays, hash slices, and iterator results. For parallel iteration, see `pfor`; for pipeline-style processing, prefer `|> e` or `|> map`. C-style `for (INIT; COND; STEP)` is also supported.

```
for my $f (glob "*.txt") { p $f }
for (1:5) { p _ * 2 }            # _ is default
my @names = qw(alice bob carol)
for (@names) { _ = uc _ }         # mutates in-place
p join ", ", @names                 # ALICE, BOB, CAROL
```

while

Loop that re-evaluates its condition before each iteration and continues as long as it is true. Commonly used for reading input line-by-line, polling, and indefinite iteration. The condition is tested in boolean context. `while` integrates naturally with the diamond operator `<>` for reading filehandles. The loop variable can be declared in the condition with `my`, scoping it to the loop body. Postfix form is also supported: `EXPR while COND;`.

```
while (my $line = <STDIN>) {
    $line |> tm |> p             # trim + print each line
}
my $i = 0
while ($i < 5) { p $i++ }    # counted loop
while (1) { last if done() }     # infinite loop with break
```

until

Loop that continues as long as its condition is *false* — the logical inverse of `while`. Useful when the termination condition is more naturally expressed as a positive assertion ("keep going until X happens"). Supports both block and postfix forms. Prefer `while` with a negated condition if `until` makes the logic harder to read.

```
my $n = 1
until ($n > 1000) { $n *= 2 }
p $n   # 1024

my $tries = 0
until (connected()) {
    $tries++
    sleep 1
}
p "connected after $tries tries"
```

do

Execute a block and return its value, or execute a file. As a block, `do { ... }` creates an expression scope — the last expression in the block is the return value, making it useful for complex initializations. As a file operation, `do "file.pl"` executes the file in the current scope and returns its last expression. Unlike `require`, `do` does not cache and re-executes on each call. `do { ... } while (COND)` creates a loop that always runs at least once.

```
my $val = do { my $x = 10
    $x ** 2 }   # 100
p $val
my $cfg = do "config.pl"               # load config
# do-while: body runs at least once
my $input
do { $input = readline(STDIN) |> tm } while ($input eq "")
```

last

Immediately exit the innermost enclosing loop (equivalent to `break` in C/Rust). Execution continues after the loop. `last LABEL` can target a labeled outer loop to break out of nested loops. Works in `for`, `foreach`, `while`, `until`, and `do-while`. Does *not* work inside `map`, `grep`, or `|>` pipeline stages — use `take`, `first`, or `take_while` for early termination in functional contexts.

```
for (1:1_000_000) {
    last if _ > 5
    p _
}   # prints 1 2 3 4 5

OUTER: for my $i (1:10) {
    for my $j (1:10) {
        last OUTER if $i * $j > 50   # break both loops
    }
}
```

For pipeline early-exit: `1:1000 |> take_while { _ < 50 } |> e p`.

next

Skip the rest of the current loop iteration and jump to the next one. The loop condition (for `while`/`until`) or the next element (for `for`/`foreach`) is evaluated immediately. Like `last`, `next` supports labeled loops with `next LABEL` for skipping in nested loops. This is the primary tool for filtering within imperative loops. In stryke, consider `grep`/`filter` or `|> reject` for functional-style filtering instead.

```
for (1:10) {
    next if _ % 2        # skip odds
    p _
}   # 2 4 6 8 10

for my $file (glob "*") {
    next unless -f $file   # skip non-files
    next if $file =~ /^\./  # skip hidden
    p $file
}
```

redo

Restart the current loop iteration from the top of the loop body *without* re-evaluating the loop condition or advancing to the next element. The loop variable retains its current value. This is a niche but powerful tool for retry logic within loops — when an iteration fails, `redo` lets you try again with the same input. Use sparingly, as it can create infinite loops if the retry condition never resolves. Always pair with a guard or counter. For automated retry with backoff, prefer `retry { ... } times => N, backoff => 'exponential'`.

```
for my $url (@urls) {
    my $body = eval { fetch($url) }
    if ($@) {
        warn "retry $url: $@"
        sleep 1
        redo   # try same URL again
    }
    p length($body)
}
```

continue

A block attached to a `for`/`foreach`/`while` loop that executes after each iteration, even when `next` is called. Analogous to the increment expression in a C-style `for` loop. The `continue` block does *not* run when `last` or `redo` is used. Useful for unconditional per-iteration bookkeeping like incrementing counters, logging progress, or flushing buffers. Rarely used but fully supported in stryke.

```
my $count = 0
for my $item (@work) {
    next if $item->{skip}
    process($item)
} continue {
    $count++
    p "processed $count so far" if $count % 100 == 0
}
```

given

A switch-like construct that evaluates an expression and dispatches to `when` blocks via smartmatch semantics. The topic variable `_` is set to the `given` expression for the duration of the block. Each `when` clause is tested in order; the first match executes its block and control passes out of the `given` (implicit break). A `default` block handles the no-match case. In stryke, prefer the `match` keyword for new code — it offers pattern destructuring, typed patterns, array/hash shape matching, and `if` guards that `given`/`when` cannot express.

```
given ($cmd) {
    when ("start")   { p "starting up" }
    when ("stop")    { p "shutting down" }
    when (/^re/)     { p "restarting" }
    default          { p "unknown: $cmd" }
}
```

See `match` for stryke-native pattern matching with destructuring.

when

A case clause inside a `given` block. The expression is matched against the topic `_` using smartmatch semantics: strings match exactly, regexes match against `_`, arrayrefs check membership, coderefs are called with `_` as argument, and numbers compare numerically. When a `when` clause matches, its block executes and control exits the enclosing `given` (implicit break). Multiple `when` clauses are tried in order until one matches.

```
given ($val) {
    when (/^\d+$/)      { p "number" }
    when (["a","b","c"]) { p "early letter" }
    when (42)            { p "the answer" }
    default              { p "something else" }
}
```

In stryke, the `match` keyword provides more powerful pattern matching.

default

The fallback clause in a `given` block, executed when no `when` clause matched. Every `given` should have a `default` to handle unexpected values, similar to `else` in an `if` chain or the wildcard `_` arm in stryke `match`. If no `default` is present and nothing matches, execution simply continues after the `given` block. In stryke `match`, use `_ => ...` for the default arm instead.

```
given ($exit_code) {
    when (0) { p "success" }
    when (1) { p "general error" }
    when (2) { p "misuse" }
    default  { p "unknown exit code: $exit_code" }
}
```

return

Return a value from a subroutine.

```
fn pin_val($v, $lo, $hi) {
    return $lo if $v < $lo
    return $hi if $v > $hi
    $v
}
```

not

Low-precedence logical negation — returns true if the expression is false, and false if it is true. Functionally identical to `!` but binds looser than almost everything, so `not $a == $b` is `not($a == $b)` rather than `(!$a) == $b`. Useful for readable boolean conditions.

```
if (not defined $val) {
    p "val is undef"
}
my @missing = grep { not -e _ } @files
@missing |> e p
p not 0    # 1
p not 1    # (empty string)
```

Error Handling

8 topics

try

Structured exception handling that cleanly separates the happy path from error recovery. The `try` block runs the code; if it throws (via `die`), execution jumps to the `catch` block with the exception bound to the declared variable. An optional `finally` block runs unconditionally afterward — ideal for cleanup like closing filehandles or releasing locks. Unlike `eval`, `try`/`catch` is a first-class statement with proper scoping and no `$@` pollution. In stryke, `try` integrates with all exception types including string messages, hashrefs, and objects.

```
try {
    my $data = fetch_json($url)
    p $data->{name}
} catch ($e) {
    warn "request failed: $e"
    return fallback()
} finally {
    log_info("fetch attempt complete")
}
```

Prefer `try`/`catch` over `eval { }` for new code — it reads better and avoids `$@` clobbering races.

catch

The error-handling clause that follows a `try` block. When the `try` block throws an exception, execution transfers to `catch` with the exception value bound to the declared variable (e.g. `$e`). The catch variable is lexically scoped to the catch block. You can inspect the exception — it may be a string, a hashref with structured error info, or an object with methods. Multiple error types can be differentiated with `ref` or `match` inside the catch body. If the catch block itself throws, the exception propagates upward (the `finally` block still runs first, if present).

```
try { die { code => 404, msg => "not found" } }
catch ($e) {
    if (ref $e eq 'HASH') {
        p "error $e->{code}: $e->{msg}"
    } else {
        p "caught: $e"
    }
}
```

finally

A cleanup block that runs after `try`/`catch` regardless of whether an exception was thrown or not. This guarantees resource cleanup even if the `try` block throws or the `catch` block re-throws. The `finally` block cannot change the exception or the return value — it is strictly for side effects like closing filehandles, releasing locks, or logging. If `finally` itself throws, that exception replaces the original one (avoid throwing in finally). `finally` is optional — you can use `try`/`catch` without it.

```
my $fh
try {
    open $fh, '<', $path or die $!
    process(<$fh>)
} catch ($e) {
    log_error("failed: $e")
} finally {
    close $fh if $fh   # always cleanup
}
```

eval

The classic Perl exception-catching mechanism. `eval { BLOCK }` executes the block in an exception-trapping context: if the block throws (via `die`), execution continues after the `eval` with the error stored in `$@`. `eval "STRING"` compiles and executes Perl code at runtime — powerful but dangerous (code injection risk). In stryke, prefer `try`/`catch` for exception handling as it avoids the `$@` clobbering pitfalls and reads more clearly. `eval` remains useful for dynamic code evaluation and backward compatibility with Perl 5 idioms.

```
eval { die "oops" }
if ($@) { p "caught: $@" }

# Eval string — dynamic code execution
my $expr = "2 + 2"
my $result = eval $expr
p $result   # 4
```

Caveat: `$@` can be clobbered by intervening `eval`s or destructors — `try`/`catch` avoids this.

die

Raise an exception, immediately unwinding the call stack until caught by `try`/`catch` or `eval`. The argument can be a string (most common), a reference (hashref for structured errors), or an object. If uncaught, the program terminates and the message is printed to STDERR. In stryke, `die` works identically to Perl 5 and integrates with `try`/`catch`, `eval`, and the `$@` mechanism. Convention: end die messages with `\n` to suppress the automatic "at FILE line LINE" suffix, or omit it to get location info for debugging.

```
fn divide($a, $b) {
    die "division by zero\n" unless $b
    $a / $b
}

# Structured error
die { code => 400, msg => "bad request", field => "email" }

# With automatic location
die "something broke"   # prints: something broke at script.pl line 5.
```

warn

Print a warning message to STDERR without terminating the program. Behaves like `die` but only emits the message instead of throwing an exception. If the message does not end with `\n`, Perl appends the current file and line number. Warnings can be intercepted with `$SIG{__WARN__}` or suppressed with `no warnings`. In stryke, `warn` is useful for non-fatal diagnostics; for structured logging, use `log_warn` instead.

```
warn "file not found: $path" unless -e $path
warn "deprecated: use fetch_json instead\n"  # no line number

# Intercept warnings
local $SIG{__WARN__} = fn ($msg) { log_warn($msg) }
warn "redirected to logger"
```

croak

Die from the caller's perspective — the error message reports the file and line of the *caller*, not the function that called `croak`. This is the right choice for library/module functions where the error is the caller's fault (bad arguments, misuse). Without `croak`, the user sees an error pointing at library internals, which is unhelpful. In stryke, `croak` is available as a builtin without `use Carp`. For debugging deep call chains, use `confess` instead to get a full stack trace.

```
fn parse($s) {
    croak "parse: empty input" unless length $s
    croak "parse: not JSON" unless $s =~ /^[{\[]/
    json_decode($s)
}

# Error will point at the call site, not inside parse()
my $data = parse("")   # dies: "parse: empty input at caller.pl line 5"
```

confess

Die with a full stack trace from the point of the error all the way up through every caller. This is invaluable for debugging deep call chains where `die` or `croak` only show one frame. Each frame includes the package, file, and line number. In stryke, `confess` is available as a builtin without `use Carp`. Use `confess` during development for maximum diagnostic info; switch to `croak` in production-facing libraries where the trace would confuse end users.

```
fn validate($data) {
    confess "missing required field 'name'" unless $data->{name}
}
fn process($input) { validate($input) }
fn main { process({}) }   # full trace: main -> process -> validate
```

The trace output shows: `missing required field 'name' at script.pl line 2.\n\tmain::validate called at line 4\n\t...`.

AOP / Advice

7 topics

before

`before "<glob>" { ... }` — register advice that runs *before* every call to a sub whose name matches the glob pattern. Inside the body, `$INTERCEPT_NAME` holds the called sub's name and `@INTERCEPT_ARGS` holds the args.

The leading keyword only commits to advice parsing when followed by a string literal, so `before(...)` as a normal sub call still parses normally. Multiple `before` advices on the same name all fire in registration order. The advice cannot suppress the call (use `around` for that) and its return value is discarded.

```
before "fetch" { warn "calling fetch with @INTERCEPT_ARGS" }
before "log_*" { $log_count++ }     # glob: any sub starting with log_
before "*"     { trace($INTERCEPT_NAME) }   # every sub call
```

Bodies are lowered to bytecode and dispatched through the VM (`run_block_region`), so `our $x` and other compile-time name resolutions work the same inside advice as outside it. The body's final statement must be an expression (same constraint as `map { }` block lowering); a literal `for`/`while`/`if` block or a literal `return` will be rejected at firing time.

after

`after "<glob>" { ... }` — register advice that runs *after* every call to a sub whose name matches the glob pattern. The body sees the call's return in `$INTERCEPT_RESULT`, the wall-clock duration in `$INTERCEPT_MS` (millis, float) and `$INTERCEPT_US` (micros, integer), the called name in `$INTERCEPT_NAME`, and the args in `@INTERCEPT_ARGS`.

Multiple `after` advices on the same name all fire in registration order. The advice's return value is discarded — the call's return is whatever the original sub (or a matching `around`) produced.

```
after "fetch" { warn "fetch returned $INTERCEPT_RESULT in ${INTERCEPT_MS}ms" }
after "*"     { log_call($INTERCEPT_NAME, $INTERCEPT_US) }
```

around

`around "<glob>" { ... }` — register advice that *wraps* every call to a sub whose name matches the glob pattern. Use `proceed()` to invoke the original; the around block's evaluated value is the call's return (AspectJ-style).

The first matching `around` on a given name wraps; later `around` matches are skipped (mirrors zshrs `run_intercepts`). If `proceed()` is not called, the original sub never runs and the around block's value replaces it. The block can transform (`proceed() + 100`), forward (just `proceed()`), or replace (omit `proceed()` and emit a value).

```
around "expensive" {
    my $cached = cache_get($INTERCEPT_ARGS[0])
    return $cached if defined $cached
    my $r = proceed()
    cache_put($INTERCEPT_ARGS[0], $r)
    $r
}

around "flaky" { my $r = eval { proceed() }; $@ ? retry_default() : $r }
```

Recursion guard: calling the advised sub from inside its own advice runs the original directly without re-firing advice — no infinite loop.

proceed

`proceed()` — invoke the original sub with the saved args from inside an `around` advice block. Returns the original's value. Calling `proceed` outside an `around` block is a runtime error.

The re-entrancy guard ensures `proceed` runs the original directly without re-firing any matching `around` advice for the same name, so transformation chains can't infinite-loop.

```
around "target" {
    my $r = proceed()
    $r * 2     # double whatever target returned
}

around "target" { proceed() // "default" }   # forward, with fallback
```

intercept_list

`intercept_list()` — return the registered AOP advice as an array of `[id, kind, pattern]` triples (each element is an arrayref).

```
before "foo" { ... }
after  "bar*" { ... }
for my $row (intercept_list()) {
    my ($id, $kind, $pat) = @$row
    p "id=$id kind=$kind pattern=$pat"
}
```

intercept_remove

`intercept_remove($id)` — remove a single AOP advice by its id (from `intercept_list`). Returns the count removed (0 or 1).

```
before "foo" { ... }
my ($id, $kind, $pat) = @{ (intercept_list())[0] }
intercept_remove($id)
p scalar intercept_list()   # 0
```

intercept_clear

`intercept_clear()` — drop all registered AOP advice. Returns the count cleared.

```
before "a" { ... }
after  "b" { ... }
p intercept_clear()         # 2
p scalar intercept_list()   # 0
```

Declarations

10 topics

my

Declare a lexically scoped variable visible only within the enclosing block or file. `my` is the workhorse of Perl variable declarations — use it for scalars (`$`), arrays (`@`), and hashes (`%`). Variables declared with `my` are invisible outside their scope, which prevents accidental cross-scope mutation. In stryke, `my` variables participate in pipe chains and can be destructured in list context. Uninitialized `my` variables default to `undef`.

```
my $name = "world"
my @nums = 1:5
my %cfg = (debug => 1, verbose => 0)
p "hello $name"
@nums |> grep _ > 2 |> e p$1

Use `my` everywhere unless you specifically need `our` (package global), `local` (dynamic scope), or `state` (persistent).
```

our

Declare a package-global variable accessible as a lexical alias in the current scope. Unlike `my`, `our` variables are visible across the entire package and can be accessed from other packages via their fully qualified name (e.g. `$Counter::total`). Useful for package-level configuration, shared counters, or variables that need to survive across file boundaries.

In stryke, `our` variables are NOT mutex-protected. Mutating an `our` variable from inside `fan`/`pmap`/`pfor` is rejected at runtime with a directive to use `oursync` (the package-global counterpart of `mysync` — same `Arc<Mutex>` backing, keyed by `Pkg::x`).

```
package Counter
our $total = 0           # serial use only
fn bump { $total++ }
package main
Counter::bump() for 1:5
p $Counter::total        # 5

# For parallel mutation, use oursync:
package C
oursync $shared = 0
fn bump { $shared++ }
package main
fan 10000 { C::bump() }   # always exactly 10000
```

Prefer `my` for local state; `our` for cross-package read-mostly globals; `oursync` for cross-package mutation under parallelism.

local

Temporarily override a global variable's value for the duration of the current dynamic scope. When the scope exits, the original value is automatically restored. This is essential for modifying Perl's special variables (like `$/`, `$\`, `$,`) without permanently altering global state. Unlike `my` which creates a new variable, `local` saves and restores an existing global. In stryke, `local` works with all special variables and respects the same restoration semantics during exception unwinding.

```
local $/ = undef       # slurp mode
open my $fh, '<', 'data.txt' or die $!
my $body = <$fh>       # reads entire file
close $fh
# $/ restored to "\n" when scope exits
```

Common patterns: `local $/ = undef` (slurp), `local $, = ","` (join print args), `local %ENV` (temporary env).

state

Declare a persistent lexical variable that retains its value across calls to the enclosing subroutine. Unlike `my`, which reinitializes on each call, `state` initializes only once — the first time execution reaches the declaration — and preserves the value for all subsequent calls. This is perfect for counters, caches, and memoization without resorting to globals or closures over external variables. In stryke, `state` variables are per-thread when used inside `~> { }` or `fan` blocks; they are not shared across workers.

```
fn ctr {
    state $n = 0
    ++$n
}
p ctr() for 1:5   # 1 2 3 4 5
fn cache_lookup($x) {
    state %cache
    $cache{$x} //= expensive($x)
}
```

Requires `use feature 'state'` in standard Perl, but is always available in stryke.

package

Set the current package namespace for all subsequent declarations. Package names are conventionally `CamelCase` with `::` separators (e.g. `Math::Utils`). All unqualified function and `our` variable names are installed into the current package. In stryke, packages work identically to standard Perl — they provide namespace isolation but are not classes by themselves (use `struct` or `bless` for OOP). Switching packages mid-file is allowed but discouraged; prefer one package per file.

```
package Math::Utils
fn calc_fact($n) { $n <= 1 ? 1 : $n * calc_fact($n - 1) }
fn calc_fib($n) { $n < 2 ? $n : calc_fib($n - 1) + calc_fib($n - 2) }
package main
p Math::Utils::calc_fact(10)   # 3628800
```

use

Load and import a user module at compile time: `use Module qw(func);`. Stryke list utilities (`sum`, `max`, `uniq`, `reduce`, ...), error helpers (`croak`, `confess`, `carp`), and serialization (`json_encode`, `json_decode`, `ddump`) are all native bare-name builtins — no import needed.

```
use My::Project::Helpers qw(normalize_path canonicalize)

my @vals = 1:10
p sum(@vals)   # 55
p max(@vals)   # 10
p normalize_path("/tmp/../etc")
```

no

Unimport a module or pragma: `no strict 'refs';`.

```
no warnings 'experimental'
given ($x) { when (1) { p "one" } }
```

require

Load a user module at runtime: `require Module;`. JSON / YAML / TOML decoding is built in (`json_decode`, `yaml_decode`, `toml_decode`) — no module load needed.

```
require My::Plugin::Loader
my $loader = My::Plugin::Loader::init()
p $loader->status

# Built-in: no require needed
my $data = json_decode($text)
p $data->{name}
```

BEGIN

`BEGIN { }` — runs at compile time, before the rest of the program.

```
BEGIN { p "compiling..." }
p "running"
# output: compiling... then running
```

END

`END { }` — runs after the program finishes (or on exit).

```
END { p "cleanup done" }
p "main work"
# output: main work then cleanup done
```

Cluster / Distributed

4 topics

cluster

`cluster(["host1:N", "host2:M", ...])` — build an SSH-backed worker pool for distributing stryke workloads across multiple machines.

Each entry in the list is a hostname (or `user@host`) followed by a colon and the number of worker slots to allocate on that host. Under the hood, stryke opens persistent SSH multiplexed connections to each host, deploys lightweight `stryke --remote-worker` processes, and manages a work-stealing scheduler across the entire cluster. The cluster object is then passed to distributed primitives like `pmap_on` and `pflat_map_on`. Workers must have `stryke` installed and accessible on `$PATH`. If a host becomes unreachable mid-computation, its in-flight tasks are automatically re-queued to surviving hosts.

```
my $cl = cluster(["server1:8", "server2:4", "gpu-box:16"])
my @results = pmap_on $cl, { heavy_compute } @jobs

# Single-machine cluster for testing:
my $local = cluster(["localhost:4"])
```

pmap_on

`pmap_on $cluster, { BLOCK } @list` — distributed parallel map that fans work across a `cluster` of remote machines.

Elements from `@list` are serialized, shipped to remote `stryke --remote-worker` processes over SSH, executed in parallel across every worker slot in the cluster, and the results are gathered back in input order. This is the distributed equivalent of `pmap`: same interface, same order guarantee, but the thread pool spans multiple hosts instead of local cores. Use this when a single machine's CPU count is the bottleneck. The block must be self-contained — it cannot close over local file handles or database connections, since it executes in a separate process on a remote host. Large closures are serialized once and cached on each worker for the lifetime of the cluster.

```
my $cl = cluster(["host1:8", "host2:8"])
my @hashes = pmap_on $cl, { sha256(slurp) } @file_paths
my @results = pmap_on $cl, { fetch("https://api.example.com/$_") } 1:10_000
```

pflat_map_on

Distributed parallel flat-map over `cluster`.

ssh

`ssh($host, $command)` — execute a shell command on a remote host via SSH and return its stdout as a string.

This is a simple synchronous wrapper around an SSH invocation. The command is run in the remote user's default shell, and stdout is captured and returned. If the remote command exits non-zero, stryke dies with the stderr output. For bulk remote work, prefer `cluster` + `pmap_on` which manages connection pooling and parallelism automatically. `ssh` is best for one-off administrative commands, health checks, or bootstrapping a remote environment before building a cluster.

```
my $uptime = ssh("server1", "uptime")
p ssh("deploy@prod", "cat /etc/hostname")
my $free = ssh("gpu-box", "nvidia-smi --query-gpu=memory.free --format=csv,noheader")
```

Datetime

15 topics

datetime_utc

Return the current UTC date and time as an ISO 8601 string (e.g. `2026-04-15T12:30:00Z`). This is the simplest way to get a portable, unambiguous timestamp for logging, file naming, or serialization. The returned string always ends with `Z` indicating UTC, so there is no timezone ambiguity.

```
my $now = datetime_utc()
p $now                          # 2026-04-15T12:30:00Z
af("audit.log", "$now: started\n")
my %event = (ts => datetime_utc(), action => "deploy")
wj("event.json", \%event)
```

datetime_from_epoch

Convert a Unix epoch timestamp (seconds since 1970-01-01 00:00:00 UTC) into an ISO 8601 datetime string. This is useful when you have raw epoch values from `time()`, file modification times, or external APIs and need a human-readable representation. Fractional seconds are truncated.

```
my $ts = 1700000000
p datetime_from_epoch($ts)       # 2023-11-14T22:13:20Z
my $born = datetime_from_epoch(0)
p $born                          # 1970-01-01T00:00:00Z
my @epochs = (1e9, 1.5e9, 2e9)
@epochs |> e { p datetime_from_epoch }
```

datetime_strftime

Format an epoch timestamp using a `strftime`-style format string, giving full control over the output representation. The first argument is the format pattern and the second is the epoch value. Supports all standard specifiers: `%Y` (4-digit year), `%m` (month), `%d` (day), `%H` (hour), `%M` (minute), `%S` (second), `%A` (weekday name), and more.

```
my $t = time()
p datetime_strftime("%Y-%m-%d", $t)        # 2026-04-15
p datetime_strftime("%H:%M:%S", $t)        # 14:23:07
p datetime_strftime("%A, %B %d", $t)       # Wednesday, April 15
my $log_ts = datetime_strftime("%Y%m%d_%H%M%S", $t)
to_file("backup_$log_ts.sql", $data)
```

datetime_now_tz

Return the current date and time in a specified IANA timezone as a formatted string. Pass a timezone name like `America/New_York`, `Europe/London`, or `Asia/Tokyo`. This avoids manual UTC-offset arithmetic and handles daylight saving transitions correctly. Dies if the timezone name is not recognized.

```
p datetime_now_tz("America/New_York")    # 2026-04-15 08:30:00 EDT
p datetime_now_tz("Asia/Tokyo")           # 2026-04-15 21:30:00 JST
p datetime_now_tz("UTC")                  # same as datetime_utc
my @offices = ("US/Pacific", "Europe/Berlin", "Asia/Kolkata")
@offices |> e { p "$_: " . datetime_now_tz }
```

datetime_format_tz

Format an epoch timestamp in a specific IANA timezone, combining the capabilities of `datetime_strftime` and `datetime_now_tz`. This lets you render a historical or future timestamp as it would appear on the wall clock in any timezone. Handles DST transitions automatically.

```
my $epoch = 1700000000
p datetime_format_tz($epoch, "America/Chicago")
# 2023-11-14 16:13:20 CST
p datetime_format_tz($epoch, "Europe/London")
# 2023-11-14 22:13:20 GMT
p datetime_format_tz(time(), "Australia/Sydney")
```

datetime_parse_local

Parse a local datetime string (without timezone info) into a Unix epoch timestamp, interpreting it in the system's local timezone. Accepts common formats like `2026-04-15 14:30:00` or `2026-04-15T14:30:00`. Dies if the string cannot be parsed. This is the inverse of formatting with `localtime`.

```
my $epoch = datetime_parse_local("2026-04-15 14:30:00")
p $epoch                          # Unix timestamp
p datetime_from_epoch($epoch)     # back to ISO 8601
my $midnight = datetime_parse_local("2026-01-01 00:00:00")
p time() - $midnight              # seconds since New Year
```

datetime_parse_rfc3339

Parse an RFC 3339 / ISO 8601 datetime string (with timezone offset or `Z` suffix) into a Unix epoch timestamp. This is the standard format used by JSON APIs, RSS feeds, and git timestamps. Accepts strings like `2026-04-15T14:30:00Z` or `2026-04-15T14:30:00+05:30`. Dies on malformed input.

```
my $epoch = datetime_parse_rfc3339("2026-04-15T12:00:00Z")
p $epoch
my $with_tz = datetime_parse_rfc3339("2026-04-15T08:00:00-04:00")
p $epoch == $with_tz              # 1 (same instant)
# parse API response timestamps
my $created = $response->{created_at}
my $age = time() - datetime_parse_rfc3339($created)
p "created ${age}s ago"
```

datetime_add_seconds

Add (or subtract) a number of seconds to an ISO 8601 datetime string and return the resulting ISO 8601 string. This performs calendar-aware arithmetic, correctly crossing day, month, and year boundaries. Pass a negative number to subtract time. Useful for computing deadlines, expiration times, or time windows.

```
my $now = datetime_utc()
my $later = datetime_add_seconds($now, 3600)     # +1 hour
p $later
my $yesterday = datetime_add_seconds($now, -86400) # -1 day
p $yesterday
my $deadline = datetime_add_seconds($now, 7 * 86400) # +1 week
p "due by $deadline"
```

elapsed

Return the number of seconds elapsed since the stryke process started, using a monotonic clock that is immune to system clock adjustments. The short alias `el` keeps benchmarking one-liners terse. Returns a floating-point value with sub-millisecond precision. Useful for timing operations, profiling hot loops, or adding relative timestamps to log output.

```
my $t0 = el()
my @sorted = sort @big_array
my $dur = el() - $t0
p "sort took ${dur}s"
# progress logging
1:100 |> e { do_work
    log_info("step $_ at " . el() . "s") }
```

time

Return the current Unix epoch as an integer — the number of seconds since 1970-01-01 00:00:00 UTC. This is the standard wall-clock timestamp used for file times, database records, and interop with external systems. For monotonic timing of code sections, prefer `elapsed`/`el` instead since `time` can jump if the system clock is adjusted.

```
my $start = time()
sleep(2)
p time() - $start   # ~2
my $ts = time()
wj("stamp.json", { created => $ts })
p datetime_from_epoch($ts)  # human-readable form
```

times

Return the accumulated CPU times for the process as a four-element list: `($user, $system, $child_user, $child_system)`. User time is CPU spent executing your code; system time is CPU spent in kernel calls on your behalf. Child times cover subprocesses. Values are in seconds (floating point). Useful for profiling whether a script is CPU-bound or I/O-bound.

```
my ($u, $s, $cu, $cs) = times()
p "user=${u}s sys=${s}s"
# after heavy computation
my ($u2, $s2) = times()
p "used " . ($u2 - $u) . "s of CPU"
p "total CPU: " . ($u2 + $s2) . "s"
```

localtime

Convert a Unix epoch timestamp to a nine-element list of broken-down local time components: `($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst)`. Follows the Perl convention where `$mon` is 0-based (January=0) and `$year` is years since 1900. When called without arguments, uses the current time. Use `gmtime` for the UTC equivalent.

```
my @t = localtime(time())
p "$t[2]:$t[1]:$t[0]"                # HH:MM:SS
my $year = $t[5] + 1900
my $mon  = $t[4] + 1
p "$year-$mon-$t[3]"                  # YYYY-M-D
my @days = qw(Sun Mon Tue Wed Thu Fri Sat)
p $days[$t[6]]                        # day of week
```

gmtime

Convert a Unix epoch timestamp to a nine-element list of broken-down UTC time components, identical in structure to `localtime` but always in the UTC timezone. The fields are `($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst)` where `$isdst` is always 0. When called without arguments, uses the current time.

```
my @utc = gmtime(time())
my $year = $utc[5] + 1900
my $mon  = $utc[4] + 1
p sprintf("%04d-%02d-%02dT%02d:%02d:%02dZ",
    $year, $mon, @utc[3,2,1,0])
# compare local vs UTC
my @loc = localtime(time())
p "UTC hour=$utc[2] local hour=$loc[2]"
```

sleep

Pause execution for the specified number of seconds. Accepts both integer and fractional values for sub-second sleeps (e.g. `sleep 0.1` for 100ms). The process yields the CPU during the sleep, so it is safe to use in polling loops without burning cycles. Returns the unslept time (always 0 unless interrupted by a signal).

```
p "waiting..."
sleep(2)
p "done"
# polling loop
while (!-e "done.flag") {
    sleep(0.5)
}
# rate limiting
my @urls = @targets
@urls |> e { fetch
    sleep(0.1) }
```

alarm

Schedule a `SIGALRM` signal to be delivered to the process after the specified number of seconds. Calling `alarm(0)` cancels any pending alarm. Only one alarm can be active at a time — setting a new alarm replaces the previous one. Returns the number of seconds remaining on the previous alarm (or 0 if none was set). Combine with `eval` and `$SIG{ALRM}` to implement timeouts around potentially hanging operations.

```
eval {
    local $SIG{ALRM} = fn { die "timeout\n" }
    alarm(5)           # 5 second deadline
    my $data = slow_network_call()
    alarm(0)           # cancel on success
}
if ($@ =~ /timeout/) {
    log_error("operation timed out")
}
```

Math

13 topics

abs

Return the absolute value of a number, stripping any negative sign. Returns the argument unchanged if it is already non-negative. Works on both integers and floating-point numbers.

```
p abs(-42)       # 42
p abs(3.14)      # 3.14
my $diff = abs($a - $b)
p "distance: $diff"
1:5 |> map { _ - 3 } |> map abs |> e p  # 2 1 0 1 2
```

int

Truncate a floating-point number toward zero, discarding the fractional part. This is not rounding — `int(1.9)` is `1` and `int(-1.9)` is `-1`. Use `sprintf("%.0f", $n)` or `POSIX::round` for proper rounding. Commonly paired with `rand` to generate random integers.

```
p int(3.7)       # 3
p int(-3.7)      # -3
p int(rand(6)) + 1  # dice roll 1:6
1:10 |> map { _ / 3 } |> map int |> e p
```

sqrt

Return the square root of a non-negative number. Dies if the argument is negative — use `abs` first or check the sign. For the inverse operation, use `squared`/`sq` or the `**` operator.

```
p sqrt(144)         # 12
p sqrt(2)           # 1.41421356...
my $hyp = sqrt($a**2 + $b**2)  # Pythagorean theorem
1:5 |> map sqrt |> e { p sprintf("%.3f", _) }
```

squared

Return the square of a number (`N * N`). This is a stryke convenience function — clearer than writing `$n ** 2` or `$n * $n` in pipelines. The aliases `sq` and `square` are interchangeable.

```
p squared(5)        # 25
p sq(12)            # 144
1:5 |> map sq |> e p    # 1 4 9 16 25
my $hyp = sqrt(sq($a) + sq($b))  # Pythagorean theorem
```

cubed

Return the cube of a number (`N * N * N`). This is a stryke convenience function for the common `$n ** 3` operation, useful in math-heavy pipelines. The aliases `cb` and `cube` are interchangeable.

```
p cubed(3)          # 27
p cb(10)            # 1000
1:4 |> map cb |> e p  # 1 8 27 64
my $vol = cb($side)             # volume of a cube
```

expt

Raise a base to an arbitrary exponent and return the result. This is the function form of the `**` operator. Accepts integer and floating-point exponents, including negative values for reciprocals and fractional values for roots.

```
p expt(2, 10)       # 1024
p expt(27, 1/3)     # 3.0 (cube root)
p expt(10, -2)      # 0.01
1:8 |> map { expt(2, _) } |> e p  # 2 4 8 16 32 64 128 256
```

exp

Return Euler's number *e* raised to the given power. `exp(0)` is `1`, `exp(1)` is approximately `2.71828`. This is the inverse of `log`. Useful for exponential growth/decay calculations, probability distributions, and converting between logarithmic and linear scales.

```
p exp(1)            # 2.71828182845905
p exp(0)            # 1
my $growth = $initial * exp($rate * $time)
1:5 |> map exp |> e { p sprintf("%.4f", _) }
```

log

Return the natural (base-*e*) logarithm of a positive number. Dies if the argument is zero or negative. For base-10 logarithms, divide by `log(10)`. For base-2, divide by `log(2)`. This is the inverse of `exp`.

```
p log(exp(1))       # 1.0
p log(100) / log(10)  # 2.0 (log base 10)
my $bits = log($n) / log(2)  # log base 2
1:5 |> map log |> e { p sprintf("%.3f", _) }
```

sin

Return the sine of an angle given in radians. The result ranges from `-1` to `1`. For degrees, convert first: `sin($deg * 3.14159265 / 180)`. Use `atan2` to go in the reverse direction.

```
p sin(0)               # 0
p sin(3.14159265 / 2)  # 1.0
my @wave = map { sin(_ * 0.1) } 0:62
@wave |> e { p sprintf("%6.3f", _) }
```

cos

Return the cosine of an angle given in radians. The result ranges from `-1` to `1`. `cos(0)` is `1`. Like `sin`, convert degrees to radians before calling.

```
p cos(0)                # 1
p cos(3.14159265)       # -1.0
my $x = $radius * cos($theta)
my $y = $radius * sin($theta)
p "($x, $y)"
```

atan2

Return the arctangent of `Y/X` in radians, using the signs of both arguments to determine the correct quadrant. The result ranges from `-pi` to `pi`. This is the standard way to compute angles from Cartesian coordinates and is more robust than `atan(Y/X)` because it handles `X=0` correctly.

```
my $pi = atan2(0, -1)       # 3.14159265...
p atan2(1, 1)               # 0.785... (pi/4)
my $angle = atan2($dy, $dx)
p sprintf("%.1f degrees", $angle * 180 / $pi)
```

rand

Return a pseudo-random floating-point number in the range `[0, N)`. If N is omitted it defaults to `1`. The result is never exactly equal to N. For integer results, combine with `int`. Seed the generator with `srand` for reproducible sequences.

```
p rand()                # e.g. 0.7342...
p int(rand(100))        # random int 0:99
my @deck = 1:52
my @shuffled = sort { rand() <=> rand() } @deck  # poor shuffle
my $coin = rand() < 0.5 ? "heads" : "tails"
p $coin
```

srand

Seed the pseudo-random number generator used by `rand`. Calling `srand` with a specific value produces a reproducible sequence, which is useful for testing. Without arguments, Perl seeds from a platform-specific entropy source. You rarely need to call this explicitly — Perl auto-seeds on first use of `rand`.

```
srand(42)               # reproducible sequence
p int(rand(100))        # always the same value
srand(42)
p int(rand(100))        # same value again
srand()                 # re-seed from entropy
```

File System

34 topics

basename

Extract the filename component from a path, stripping all leading directory segments. The short alias `bn` keeps one-liner pipelines terse. If an optional suffix argument is provided, that suffix is also stripped from the result, which is handy for removing extensions.

```
p basename("/usr/local/bin/stryke")        # stryke
p bn("/tmp/data.csv", ".csv")           # data
"/etc/nginx/nginx.conf" |> bn |> p      # nginx.conf
```

dirname

Return the directory portion of a path, stripping the final filename component. The short alias `dn` mirrors `bn`. This is a pure string operation — it does not touch the filesystem, so it works on paths that do not exist yet. Useful for deriving output directories from input file paths.

```
p dirname("/usr/local/bin/stryke")          # /usr/local/bin
p dn("/tmp/data.csv")                   # /tmp
my $dir = dn($0)                        # directory of current script
```

fileparse

Split a path into its three logical components: the filename, the directory prefix, and a suffix that matches one of the supplied patterns. This mirrors Perl's `File::Basename::fileparse` and is the most flexible path decomposition available. When no suffix patterns are given the suffix is empty.

```
my ($name, $dir, $sfx) = fileparse("/home/user/report.txt", qr/\.txt/)
p "$dir | $name | $sfx"  # /home/user/ | report | .txt
my ($n, $d) = fileparse("./lib/Foo/Bar.pm")
p "$d$n"                # ./lib/Foo/Bar.pm
```

realpath

Resolve a path to its absolute canonical form by following all symbolic links and eliminating `.` and `..` segments. Unlike `canonpath`, this hits the filesystem and will die if any component does not exist. The short alias `rp` is convenient in pipelines. Use this when you need a guaranteed unique path for deduplication or comparison.

```
p realpath(".")                     # /home/user/project
p rp("../sibling")                  # /home/user/sibling
my $canon = rp($0)                 # absolute path of current script
"." |> rp |> p
```

canonpath

Clean up a file path by collapsing redundant separators, resolving `.` and `..` segments, and normalizing trailing slashes — all without touching the filesystem. Unlike `realpath`, this is a purely lexical operation so it works on paths that do not exist. Use it to normalize user-supplied paths before comparison or storage.

```
p canonpath("/usr/./local/../local/bin/")   # /usr/local/bin
p canonpath("a/b/../c")                     # a/c
my $clean = canonpath($ENV{HOME} . "/./docs/../docs")
p $clean
```

getcwd

Return the current working directory as an absolute path string. This calls the underlying OS `getcwd` function, so it always reflects the real directory even if the process changed directories via `chdir`. The alias `pwd` matches the familiar shell command. Often used to save and restore directory context around `chdir` calls.

```
my $orig = pwd()
chdir("/tmp")
p pwd()      # /tmp
chdir($orig)
p getcwd()   # back to original
```

which

Search the `PATH` environment variable for the first executable matching the given command name and return its absolute path, or `undef` if not found. This is the programmatic equivalent of the shell `which` command. Useful for checking tool availability before calling `system` or `exec`.

```
my $gcc = which("gcc") // die "gcc not found"
p $gcc                       # /usr/bin/gcc
if (which("rg")) {
    system("rg pattern file")
} else {
    system("grep pattern file")
}
```

glob

Expand a zsh-style glob pattern against the filesystem and return a list of matching paths. World-first: stryke supports every zsh glob qualifier from `man zshexpn` _Filename Generation > Glob Qualifiers_ (no other scripting language does this). Backed by `zsh::glob` from zshrs — single source of truth, zero stryke-side reimplementation. Patterns: `*`, `?`, `[abc]`, `{a,b}`, `**` recursive, plus the `(...)` qualifier suffix.

Qualifier reference (full set)

File type: `(/)` dir · `(.)` regular file · `(@)` symlink · `(=)` socket · `(p)` FIFO · `(%b)` block dev · `(%c)` char dev · `(%)` any device · `(*)` executable file.

Permission: `(r)` `(w)` `(x)` EUID r/w/x · `(R)` `(W)` `(X)` world · `(A)` `(I)` `(E)` group · `(s)` setuid · `(S)` setgid · `(t)` sticky · `(f<bits>)` exact mode-bit match (e.g. `(f644)`).

Ownership / device: `(U)` owned by EUID · `(G)` owned by EGID · `(u<N>)` uid N · `(g<N>)` gid N · `(d<N>)` match by device number.

Numeric (each accepts `+N` greater, `-N` less, `N` equal): `(L[unit]±N)` size — `p`/`k`/`m`/`g`/`t` units, default bytes · `(l±N)` link count · `(a[unit]±N)` atime · `(m[unit]±N)` mtime · `(c[unit]±N)` ctime — time units `s`/`m`/`h`/`d`/`w`/`M`.

Sort: `(on)` `(oL)` `(ol)` `(oa)` `(om)` `(oc)` `(od)` ascending; capital `O*` descending; `(oN)` no sort.

Selection: `([N])` Nth match · `([N,M])` slice. Combine with sort: `(om[1])` newest single match.

Flags: `(N)` NULL_GLOB — empty list on no match · `(D)` include dotfiles · `(F)` non-empty directory · `(M)` mark-dirs (append `/`) · `(T)` list-types (append type char) · `(n)` numeric sort.

Eval / functions: `(e'CMD')` shell-eval predicate · `(+func)` function-as-test (predicate sub).

Join words: `(P…)` prefix words · `(Q…)` postfix words around each match.

Colon modifiers: `(:s/PAT/REPL/)` sed substitution on each path · `(:e)` extension · `(:r)` strip extension · `(:t)` tail · `(:h)` head · all of zsh's history-style modifiers.

Combinators: prefix `^` negates · prefix `-` toggles symlink-following · `,` separates OR alternatives · chained qualifiers AND.

```
my @scripts = glob("*.pl")
my @all_rs = glob("src/**/*.rs")
my @dirs = glob("**(/)")           # directories only, recursive
my @big = glob("**(L+1024)")       # files larger than 1024 bytes
my @recent = glob("**(om[1])")     # most recently modified, take 1
my @safe = glob("missing*(N)")     # null-glob, never errors
my @mode = glob("**(f644)")        # files with exactly mode 644
my @stems = glob("*.txt(:r)")      # all .txt files, extension stripped
```

glob_match

Test whether a string matches a shell-style glob pattern. Returns true (1) on match, false (empty string) otherwise. Supports `*`, `?`, `[abc]`, and `{a,b}`. This is a pure string match — it does not read the filesystem, so it works for filtering lists of paths you already have. Note: zsh glob qualifiers like `(/)` or `(.)` are stat-based filters that only apply at filesystem-expansion time (`glob`/`glob_par`/`slurp`); `glob_match` ignores any trailing qualifier.

```
p glob_match("*.pl", "script.pl")        # 1
p glob_match("*.pl", "script.py")        # (empty)
my @perl = grep { glob_match("*.{pl,pm}", _) } @files
@perl |> e p
```

copy

Copy a file from a source path to a destination path. The destination can be a file path or a directory (in which case the source filename is preserved). Dies on failure. This is the programmatic equivalent of `cp` and avoids shelling out. Metadata such as permissions is preserved where possible.

```
copy("src/config.yaml", "/tmp/config.yaml")
copy("report.pdf", "/backup/")
my $tmp = tf()
copy($0, $tmp)  # back up the current script
p slurp($tmp)
```

move

Move or rename a file from source to destination. If the source and destination are on the same filesystem this is an atomic rename; otherwise it falls back to copy-then-delete. Dies on failure. The short alias `mv` mirrors the shell command.

```
move("draft.txt", "final.txt")         # rename in place
mv("output.csv", "/archive/output.csv") # move across dirs
my $tmp = tf()
spurt($tmp, "data")
mv($tmp, "data.txt")
```

rename

Rename a file or directory from an old name to a new name. This is an atomic operation on the same filesystem. If the destination already exists it is silently replaced. Dies on failure. Unlike `move`/`mv`, this does not fall back to copy-then-delete across filesystems.

```
rename("draft.md", "final.md")
rename("output", "output_v2")          # works on dirs too
my $bak = "config.yaml.bak"
rename("config.yaml", $bak)
spurt("config.yaml", $new_config)
```

unlink

Delete one or more files from the filesystem. Returns the number of files successfully removed. Does not remove directories — use `rmdir` for those. Dies on permission errors. Accepts a list of paths, making it convenient for batch cleanup.

```
unlink("temp.log")
my $n = unlink("a.tmp", "b.tmp", "c.tmp")
p "removed $n files"
my @old = glob("*.bak")
unlink(@old)
```

mkdir

Create a directory at the given path. An optional second argument specifies the permission mode as an octal number (default `0777`, modified by the current `umask`). Dies if the directory cannot be created. Only creates one level — use `make_path` or shell out for recursive creation.

```
mkdir("output")
mkdir("/tmp/secure", 0700)
my $dir = tdr() . "/sub"
mkdir($dir)
p -d $dir    # 1
```

rmdir

Remove an empty directory. Dies if the directory does not exist, is not empty, or cannot be removed due to permissions. This only removes a single directory — it will not recursively delete contents. Remove files with `unlink` first, then call `rmdir`.

```
mkdir("scratch")
rmdir("scratch")
p -d "scratch"   # (empty, dir is gone)
unlink("tmp/file.txt")
rmdir("tmp")
```

chmod

Change the permission bits of one or more files. The mode is specified as an octal number. Returns the number of files successfully changed. Does not follow symlinks on some platforms. Use `stat` to read the current mode before modifying.

```
chmod(0755, "script.pl")
chmod(0644, "config.yaml", "data.json")
my $n = chmod(0600, glob("*.key"))
p "secured $n key files"
```

chown

Change the owner and group of one or more files, specified as numeric UID and GID. Pass `-1` for either to leave it unchanged. Returns the number of files successfully changed. Typically requires root privileges.

```
chown(1000, 1000, "app.log")
chown(-1, 100, "shared.txt")     # change group only
my $uid = (getpwnam("deploy"))[2]
chown($uid, -1, "release.tar")
```

chdir

Change the current working directory of the process. Dies if the directory does not exist or is not accessible. This affects all subsequent relative path operations. Pair with `getcwd`/`pwd` to save and restore the original directory.

```
my $orig = pwd()
chdir("/tmp")
spurt("scratch.txt", "hello")
chdir($orig)                # return to original
```

stat

Return a 13-element list of file status information for a path or filehandle: `($dev, $ino, $mode, $nlink, $uid, $gid, $rdev, $size, $atime, $mtime, $ctime, $blksize, $blocks)`. This calls the POSIX `stat` system call. Use it to check file size,

modification time, permissions, and other metadata without reading the file.

```
my @st = stat("data.bin")
p "size: $st[7] bytes"
p "modified: " . datetime_from_epoch($st[9])
my ($mode) = (stat($0))[2]
p sprintf("perms: %04o", $mode & 07777)
```

link

Create a hard link — a new directory entry pointing to the same underlying inode as the original file. Both names are indistinguishable and share the same data; removing one does not affect the other. Hard links cannot cross filesystem boundaries and typically cannot link directories.

```
link("data.csv", "data_backup.csv")
my @st1 = stat("data.csv")
my @st2 = stat("data_backup.csv")
p $st1[1] == $st2[1]   # 1 — same inode
```

symlink

Create a symbolic (soft) link that points to a target path. Unlike hard links, symlinks can cross filesystems and can point to directories. The link stores the target as a string, so it can dangle if the target is later removed. Use `readlink` to inspect where a symlink points.

```
symlink("/usr/local/bin/stryke", "pe_link")
p readlink("pe_link")       # /usr/local/bin/stryke
p -l "pe_link"              # 1 (is a symlink)
symlink("../lib", "lib_link")  # relative target
```

readlink

Return the target path that a symbolic link points to, without following the link further. Returns `undef` if the path is not a symlink. This is useful for inspecting symlink chains, verifying link targets, or resolving one level of indirection at a time.

```
symlink("real.conf", "link.conf")
my $target = readlink("link.conf")
p $target                    # real.conf
if (defined readlink($path)) {
    p "$path is a symlink"
}
```

utime

Set the access time and modification time of one or more files. Times are specified as epoch seconds. Pass `undef` for either time to set it to the current time. Returns the number of files successfully updated. Useful for cache invalidation, build systems, or preserving timestamps after transformations.

```
utime(time(), time(), "output.txt")     # touch to now
my $epoch = 1700000000
utime($epoch, $epoch, @files)           # backdate files
utime(undef, undef, "marker.flag")      # equivalent of touch
```

umask

Get or set the file creation mask, which controls the default permissions for newly created files and directories. When called with an argument, sets the new mask and returns the previous one. When called without arguments, returns the current mask. The mask is subtracted from the requested permissions in `mkdir`, `open`, etc.

```
my $old = umask(0077)         # restrict: owner-only
mkdir("private")               # created with 0700
umask($old)                    # restore previous mask
p sprintf("%04o", umask())     # print current mask
```

uname

Return system identification as a five-element list: `($sysname, $nodename, $release, $version, $machine)`. This calls the POSIX `uname` system call and is useful for platform-specific logic, logging system info, or generating diagnostic reports without shelling out.

```
my ($sys, $node, $rel, $ver, $arch) = uname()
p "$sys $rel ($arch)"           # Linux 6.1.0 (x86_64)
if ($sys eq "Darwin") {
    p "running on macOS"
}
log_info("host: $node, kernel: $rel")
```

gethostname

Return the hostname of the current machine as a string. This calls the POSIX `gethostname` system call. The short alias `hn` is useful in log prefixes, temp-file naming, or distributed-system identifiers where you need to tag output by machine.

```
p gethostname()                          # myhost.local
my $log_prefix = hn() . ":" . $$        # myhost.local:12345
log_info("running on " . hn())
```

opendir

Open a directory handle for reading its entries. Returns a directory handle that can be passed to `readdir`, `seekdir`, `telldir`, `rewinddir`, and `closedir`. Dies if the directory does not exist or cannot be opened. For most use cases `glob` or `readdir` with a path is simpler.

```
opendir(my $dh, "/tmp") or die "cannot open: $!"
my @entries = readdir($dh)
closedir($dh)
@entries |> grep { _ !~ /^\./ } |> e p  # skip dotfiles
```

readdir

Read entries from a directory handle opened with `opendir`. In list context, returns all remaining entries. In scalar context, returns the next single entry or `undef` when exhausted. Entries include `.` and `..` so you typically filter them out.

```
opendir(my $dh, ".") or die $!
while (my $entry = readdir($dh)) {
    next if $entry =~ /^\./
    p $entry
}
closedir($dh)
```

closedir

Close a directory handle previously opened with `opendir`, releasing the underlying OS resource. While directory handles are closed automatically when they go out of scope, explicit `closedir` is good practice in long-running programs or loops that open many directories.

```
opendir(my $dh, "/var/log") or die $!
my @logs = readdir($dh)
closedir($dh)
@logs |> grep { /\.log$/ } |> e p
```

seekdir

Set the current position in a directory handle to a location previously obtained from `telldir`. This allows you to revisit directory entries without closing and reopening the handle. Rarely needed in practice, but useful for multi-pass directory scanning.

```
opendir(my $dh, ".") or die $!
my $pos = telldir($dh)
my @first_pass = readdir($dh)
seekdir($dh, $pos)              # rewind to saved position
my @second_pass = readdir($dh)
closedir($dh)
```

telldir

Return the current read position within a directory handle as an opaque integer. The returned value can be passed to `seekdir` to return to that position later. This is the directory-handle equivalent of `tell` for file handles.

```
opendir(my $dh, "/tmp") or die $!
readdir($dh)                  # skip .
readdir($dh)                  # skip ..
my $pos = telldir($dh)        # save position after . and ..
my @real = readdir($dh)
seekdir($dh, $pos)            # go back
```

rewinddir

Reset a directory handle back to the beginning so that the next `readdir` returns the first entry again. This is equivalent to closing and reopening the directory but more efficient. Useful when you need to iterate a directory multiple times.

```
opendir(my $dh, "src") or die $!
my $count = scalar readdir($dh)
rewinddir($dh)
my @entries = readdir($dh)
closedir($dh)
p "$count entries"
```

du

`du` (alias `dir_size`) — compute the total size of a directory tree in bytes, recursively walking all files.

```
my $bytes = du("/usr/local")
p "$bytes bytes"
```

du_tree

`du_tree` (alias `dir_sizes`) — return directory sizes as an array of hashrefs `{path, size}`, sorted descending by size. Each entry is an immediate child directory.

```
my @dirs = du_tree("/usr/local")
for (@dirs) { p "_->{path}: _->{size}" }
```

Process

21 topics

system

Execute a command in a subshell and wait for it to complete, returning the exit status. A return value of `0` means success; non-zero indicates failure. The exit status is encoded as `$? >> 8` for the actual exit code. Use backticks or `capture` if you need the command's output.

```
my $rc = system("make", "test")
p "exit code: " . ($rc >> 8)
system("cp data.csv /backup/") == 0 or die "copy failed"
if (system("which rg >/dev/null 2>&1") == 0) {
    p "ripgrep is installed"
}
```

exec

Replace the current process image entirely with a new command. This never returns on success — the new program takes over. If `exec` fails (command not found, permission denied) it returns false and execution continues. Use `system` instead if you want to run a command and keep the current process alive.

```
exec("ls", "-la", "/tmp") or die "exec failed: $!"
# code here only runs if exec fails

# common fork+exec pattern
if (fork() == 0) {
    exec("worker", "--daemon")
}
```

fork

Fork the current process, creating a child that is an exact copy of the parent. Returns the child's PID to the parent process and `0` to the child, allowing each side to branch. Returns `undef` on failure. Always pair with `wait`/`waitpid` to reap the child and avoid zombies.

```
my $pid = fork()
if ($pid == 0) {
    p "child $$"
    exit(0)
}
p "parent $$, child is $pid"
waitpid($pid, 0)
```

wait

Wait for any child process to terminate and return its PID. The exit status is stored in `$?`. If there are no child processes, returns `-1`. This is the simplest reaping function — use `waitpid` when you need to wait for a specific child or use non-blocking flags.

```
for (1:3) {
    fork() or do { sleep(1)
    exit(0) }
}
while ((my $pid = wait()) != -1) {
    p "child $pid exited with " . ($? >> 8)
}
```

waitpid

Wait for a specific child process identified by PID to change state. The flags argument controls behavior — use `0` for blocking wait, or `WNOHANG` for non-blocking (returns `0` if child is still running). Returns the PID on success, `-1` if the child does not exist. Exit status is in `$?`.

```
my $pid = fork() // die "fork: $!"
if ($pid == 0) { sleep(2)
    exit(42) }
waitpid($pid, 0)
p "child exited: " . ($? >> 8)   # 42

# non-blocking poll
while (waitpid($pid, WNOHANG) == 0) {
    p "still running..."
    sleep(1)
}
```

kill

Send a signal to one or more processes by PID. The signal can be specified as a number (`9`) or a name (`"TERM"`). Sending signal `0` tests whether the process exists without actually sending anything. Returns the number of processes successfully signaled.

```
kill("TERM", $child_pid)
kill(9, @worker_pids)               # SIGKILL
if (kill(0, $pid)) {
    p "process $pid is alive"
}
my $n = kill("HUP", @daemons)
p "signaled $n processes"
```

exit

Terminate the program immediately with the given exit status code. An exit code of `0` conventionally means success; any non-zero value indicates an error. `END` blocks and object destructors are still run. Use `POSIX::_exit` to skip cleanup entirely.

```
exit(0)                # success
exit(1) if $error      # failure

# conditional exit in a pipeline
my $ok = system("make test")
exit($ok >> 8) if $ok
```

getlogin

Return the login name of the user who owns the current terminal session. This reads from the system's utmp/utmpx database and may return `undef` for processes without a controlling terminal (cron jobs, daemons). For a more reliable alternative, use `getpwuid($<)` which looks up the effective UID.

```
my $user = getlogin() // (getpwuid($<))[0]
p "running as $user"
log_info("session started by " . getlogin())
```

getpwnam

Look up user account information by username string. Returns the same 9-element list as `getpwuid`: `($name, $passwd, $uid, $gid, $quota, $comment, $gcos, $dir, $shell)`. Returns empty list if the user does not exist. Useful for resolving a username to a UID before calling `chown`.

```
my @info = getpwnam("deploy")
p "uid=$info[2] home=$info[7]"
my $uid = (getpwnam("www-data"))[2]
chown($uid, -1, "public/index.html")
```

getpwuid

Look up user account information by numeric UID. Returns `($name, $passwd, $uid, $gid, $quota, $comment, $gcos, $dir, $shell)` on success, or empty list if the UID does not exist. This is the reliable way to map a UID to a username and home directory.

```
my ($name, undef, undef, undef, undef, undef, undef, $home) = getpwuid($<)
p "user: $name, home: $home"
my $root_shell = (getpwuid(0))[8]
p $root_shell   # /bin/bash or /bin/zsh
```

getpwent

Read the next entry from the system password database, iterating through all user accounts. Returns `($name, $passwd, $uid, $gid, $quota, $comment, $gcos, $dir, $shell)` for each user, or empty list when exhausted. Call `setpwent` to rewind and `endpwent` to close.

```
while (my @pw = getpwent()) {
    p "$pw[0]: uid=$pw[2] home=$pw[7]"
}
endpwent()
```

getgrgid

Look up group information by numeric GID. Returns `($name, $passwd, $gid, $members)` where members is a space-separated string of usernames belonging to the group. Returns empty list if the GID does not exist.

```
my ($name, undef, undef, $members) = getgrgid(0)
p "group $name: $members"
my $gname = (getgrgid((stat("file.txt"))[5]))[0]
p "file group: $gname"
```

getgrnam

Look up group information by group name string. Returns `($name, $passwd, $gid, $members)`. Useful for resolving a group name to a GID before calling `chown`, or for checking group membership.

```
my ($name, undef, $gid, $members) = getgrnam("staff")
p "gid=$gid members=$members"
chown(-1, $gid, "shared_dir")
if ((getgrnam("admin"))[3] =~ /\b$user\b/) {
    p "$user is an admin"
}
```

getgrent

Read the next entry from the system group database, iterating through all groups. Returns `($name, $passwd, $gid, $members)` for each group, or empty list when exhausted. The members field is a space-separated string of usernames. Call `endgrent` when done.

```
while (my @gr = getgrent()) {
    p "$gr[0]: gid=$gr[2] members=$gr[3]"
}
endgrent()
```

getppid

Return the process ID of the parent process. This is useful for detecting whether the process has been orphaned (parent PID becomes 1 on Unix when the original parent exits), or for logging the process hierarchy. Always returns a valid PID.

```
p "my pid: $$, parent: " . getppid()
if (getppid() == 1) {
    log_warn("parent process has exited, we are orphaned")
}
```

getpgrp

Return the process group ID of the current process (or of the specified PID). Processes in the same group receive signals together — for example, Ctrl-C sends SIGINT to the entire foreground process group. Use `setpgrp` to move a process into a different group.

```
p "process group: " . getpgrp()
my $pg = getpgrp($$)
p $pg == $$ ? "group leader" : "group member"
```

setpgrp

Set the process group ID of a process. Call `setpgrp(0, 0)` to make the current process a new group leader, which is useful for daemonization or isolating a subprocess from the terminal's signal group. Takes `(PID, PGID)` — use `0` for the current process.

```
setpgrp(0, 0)   # become process group leader
if (fork() == 0) {
    setpgrp(0, 0)  # child gets its own group
    exec("worker")
}
```

getpriority

Get the scheduling priority (nice value) of a process, process group, or user. The `which` argument selects the target type: `PRIO_PROCESS`, `PRIO_PGRP`, or `PRIO_USER`. Lower values mean higher priority. The default nice value is `0`; range is typically `-20` to `19`.

```
my $nice = getpriority(0, $$)    # PRIO_PROCESS, current PID
p "nice value: $nice"
my $user_prio = getpriority(2, $<)  # PRIO_USER, current user
p "user priority: $user_prio"
```

setpriority

Set the scheduling priority (nice value) of a process, process group, or user. Lowering the nice value (higher priority) typically requires root privileges. Raising it (lower priority) is always allowed. Use this to deprioritize background batch work or boost latency-sensitive tasks.

```
setpriority(0, $$, 10)   # lower priority for batch work
if (fork() == 0) {
    setpriority(0, 0, 19)  # lowest priority for child
    exec("batch-job")
}
```

syscall

Invoke a raw system call by its numeric identifier, passing arguments directly to the kernel. This is an escape hatch for system calls that have no Perl wrapper. The call number is platform-specific and the arguments must be correctly typed (integers or string buffers). Use with caution — incorrect arguments can crash the process.

```
# SYS_getpid on Linux x86_64 is 39
my $pid = syscall(39)
p $pid                    # same as $$
# SYS_sync on Linux is 162
syscall(162)              # flush filesystem caches
```

process_list

`process_list` (aliases `ps`, `procs`) — list running processes as an array of hashrefs with `{pid, name, uid}` keys.

```
my @procs = ps()
for (@procs) { p "_->{pid} _->{name}" }
```

Pack / Binary

4 topics

pack

Convert a list of values into a binary string according to a template. Each template character specifies how one value is encoded: `N` for 32-bit big-endian unsigned, `n` for 16-bit, `a` for raw bytes, `Z` for null-terminated string, etc. This is essential for constructing binary protocols, file formats, and `sockaddr` structures.

```
my $bin = pack("NnA4", 0xDEADBEEF, 8080, "test")
p length($bin)                # 10 bytes
my $header = pack("A8 N N", "MAGIC01\0", 1, 42)
spurt("data.bin", $header)
```

unpack

Decode a binary string into a list of values according to a template, performing the inverse of `pack`. The template characters must match how the data was packed. Use this for parsing binary file formats, network protocol headers, or any structured binary data.

```
my ($magic, $version, $count) = unpack("A8 N N", $header)
p "v$version, $count records"
my ($port, $addr) = unpack("n a4", $sockaddr)
p inet_ntoa($addr) . ":$port"
```

unpack_first

`unpack_first(FMT, STR)` — decode the binary string and return the first decoded element (or `undef` if the format yields nothing). The stryke spelling of Perl's `scalar unpack(FMT, STR)` for `--no-interop` mode where `scalar` is rejected at parse time. Aliases: `unpack1`, `up1`. Equivalent to `head(unpack(FMT, STR))` but spelled as a single verb.

```
my $first_byte = unpack_first("C", $bytes)
my $port       = unpack1("n", $sockaddr)
my $magic_word = up1("N", $header)
```

vec

Treat a string as a bit vector and get or set individual elements at a specified bit width. The first argument is the string, the second is the element offset, and the third is the bit width (1, 2, 4, 8, 16, or 32). As an lvalue, `vec` modifies the string in place. Useful for compact boolean arrays and bitmap manipulation.

```
my $bits = ""
vec($bits, 0, 1) = 1   # set bit 0
vec($bits, 7, 1) = 1   # set bit 7
p vec($bits, 0, 1)     # 1
p vec($bits, 3, 1)     # 0
p unpack("B8", $bits)  # 10000001
```

Logging

7 topics

log_info

Log a message at INFO level to stderr with a timestamp prefix. INFO is the default visible level and is appropriate for normal operational messages — startup notices, progress milestones, summary statistics. Messages are suppressed if the current log level is set higher than INFO.

```
log_info("server started on port $port")
my @rows = rl("data.csv")
log_info("loaded " . scalar(@rows) . " rows")
1:5 |> e { log_info("processing item $_") }
```

log_warn

Log a message at WARN level to stderr. Warnings indicate unexpected but recoverable situations — missing optional config, deprecated usage, slow operations. WARN messages appear at the default log level and are visually distinct from INFO in structured log output.

```
log_warn("config file not found, using defaults")
log_warn("query took ${elapsed}s, exceeds threshold")
unless (-e $path) {
    log_warn("$path missing, skipping")
}
```

log_error

Log a message at ERROR level to stderr. Use this for failures that prevent an operation from completing but do not necessarily terminate the program — failed network requests, invalid input, permission errors. ERROR is always visible regardless of log level.

```
log_error("failed to connect to $host: $!")
eval { rj("bad.json") }
log_error("parse failed: $@") if $@
log_error("missing required field 'name'")
```

log_debug

Log a message at DEBUG level to stderr. Debug messages are hidden by default and only appear when the log level is lowered to DEBUG or TRACE via `log_level`. Use for detailed internal state that helps during development — variable dumps, branch decisions, intermediate values.

```
log_level("debug")
log_debug("cache key: $key")
my $result = compute($x)
log_debug("compute($x) => $result")
@items |> e { log_debug("item: $_") }
```

log_trace

Log a message at TRACE level to stderr. This is the most verbose level, producing very fine-grained output — loop iterations, function entry/exit, raw payloads. Only visible when `log_level("trace")` is set. Use sparingly in production code; primarily for deep debugging sessions.

```
log_level("trace")
fn process($x) {
    log_trace("entering process($x)")
    my $r = $x * 2
    log_trace("leaving process => $r")
    $r
}
1:3 |> map process |> e p
```

log_json

Emit a structured JSON log line to stderr containing the message plus any additional key-value metadata. This is designed for machine-parseable logging pipelines — centralized log collectors, JSON-based monitoring, or `jq`-friendly output. Each call emits exactly one JSON object per line.

```
log_json("request", method => "GET", path => "/api")
log_json("metric", name => "latency_ms", value => 42)
log_json("error", msg => $@, file => $0)$1

Note: all values are serialized as JSON strings.
```

log_level

Get or set the current minimum log level. When called with no arguments, returns the current level as a string. When called with a level name, sets it for all subsequent log calls. Valid levels from most to least verbose: `trace`, `debug`, `info`, `warn`, `error`. The default level is `info`.

```
p log_level()          # info
log_level("debug")     # enable debug output
log_debug("now visible")
log_level("error")     # suppress everything below error
log_info("hidden")     # not printed
```

Charts (SVG)

19 topics

scatter_svg

scatter_svg (alias scatter_plot) — generate an SVG scatter plot. Args: xs, ys [, title]. Dark theme, auto-scaled axes.

```
scatter_svg([1,2,3,4], [1,4,9,16], "Squares") |> to_file("scatter.svg")
```

line_svg

line_svg (alias line_plot) — generate an SVG line plot. Args: xs, ys [, title].

```
my @x = map { _ * 0.1 } 0:100
my @y = map { sin(_) } @x
line_svg(\@x, \@y, "Sine") |> to_file("sine.svg")
```

plot_svg

plot_svg — SVG line plot with auto X axis (0..n-1). Args: ys [, title].

```
plot_svg([map { _ ** 2 } 0:50], "Parabola") |> to_file("plot.svg")
```

hist_svg

hist_svg (alias histogram_svg) — SVG histogram with auto-binning (sqrt rule). Args: data [, bins, title].

```
rnorm(1000) |> hist_svg(30, "Normal") |> to_file("hist.svg")
```

boxplot_svg

boxplot_svg (alias box_plot) — SVG box-and-whisker plot with IQR whiskers and outlier detection. Args: groups [, title]. Groups is array of arrays.

```
boxplot_svg([[1,2,3,4,5], [3,4,5,6,20]], "Compare") |> to_file("box.svg")
```

bar_svg

bar_svg (alias barchart_svg) — SVG bar chart with labeled bars and value annotations. Args: labels, values [, title].

```
bar_svg(["Rust","Go","C++"], [45,30,25], "Languages") |> to_file("bar.svg")
```

pie_svg

pie_svg (alias pie_chart) — SVG pie chart with percentage labels. Args: labels, values [, title].

```
pie_svg(["A","B","C"], [50,30,20]) |> to_file("pie.svg")
```

heatmap_svg

heatmap_svg (alias heatmap) — SVG heatmap with blue-cyan-yellow-red colormap. Args: matrix [, title].

```
heatmap_svg(cor_mat($data), "Correlation") |> to_file("heat.svg")
```

donut_svg

donut_svg (alias donut) — SVG donut chart with percentage labels. Accepts labels+values or a hashref.

```
donut_svg(["A","B","C"], [50,30,20], "Share") |> to_file("donut.svg")
{Rust => 45, Go => 30} |> donut("Languages") |> to_file("donut.svg")
```

area_svg

area_svg (alias area_chart) — SVG filled area chart. Args: xs, ys [, title].

```
my @x = 0:20
my @y = map { sin(_ * 0.3) } @x
area_svg(\@x, \@y, "Wave") |> to_file("area.svg")
```

hbar_svg

hbar_svg (alias hbar) — SVG horizontal bar chart. Accepts labels+values or a hashref.

```
hbar_svg(["Rust","Go","C++"], [45,30,25], "Speed") |> to_file("hbar.svg")
{alpha => 10, beta => 20} |> hbar |> to_file("hbar.svg")
```

radar_svg

radar_svg (aliases radar, spider) — SVG radar/spider chart. Args: labels, values [, title].

```
radar_svg(["Atk","Def","Spd","HP","MP"], [8,6,9,5,7], "Stats") |> to_file("radar.svg")
```

candlestick_svg

candlestick_svg (aliases candlestick, ohlc) — SVG candlestick OHLC chart. Args: array of [open,high,low,close] [, title].

```
candlestick_svg([[100,110,95,105],[105,115,100,112]], "Stock") |> to_file("ohlc.svg")
```

violin_svg

violin_svg (alias violin) — SVG violin plot from array of arrays showing distribution shape.

```
violin_svg([rnorm(200), rnorm(200, 2)], "Distributions") |> to_file("violin.svg")
```

cor_heatmap

cor_heatmap (alias cor_matrix_svg) — SVG correlation matrix heatmap. Computes pairwise correlations and renders as heatmap.

```
cor_heatmap([rnorm(100), rnorm(100), rnorm(100)], "Correlations") |> to_file("cor.svg")
```

stacked_bar_svg

stacked_bar_svg (alias stacked_bar) — SVG stacked bar chart. Args: labels, series (array of arrays) [, title].

```
stacked_bar_svg(["Q1","Q2","Q3"], [[10,20,30],[5,15,25]], "Revenue") |> to_file("stacked.svg")
```

wordcloud_svg

`wordcloud_svg` (aliases `wordcloud`, `wcloud`) — SVG word cloud from frequency hashref. Word size scales with frequency.

```
{rust => 50, go => 30, python => 40} |> wordcloud("Languages") |> to_file("cloud.svg")
```

treemap_svg

`treemap_svg` (alias `treemap`) — SVG treemap from frequency hashref. Area proportional to values.

```
{src => 500, tests => 200, docs => 100} |> treemap("Codebase") |> to_file("tree.svg")
```

preview

`preview` (alias `pvw`) — wrap SVG/HTML content in a cyberpunk-styled page and open in the default browser.

```
scatter_svg([1,2,3], [1,4,9]) |> preview
bar_svg(["A","B"], [10,20]) |> pvw
```

Audio

4 topics

audio_convert

`audio_convert` (alias `aconv`) — convert audio files between WAV, FLAC, AIFF, and MP3 formats.

```
audio_convert("song.flac", "song.mp3")
aconv("input.wav", "output.mp3")
```

audio_info

`audio_info` (alias `ainfo`) — get audio file metadata (duration, sample rate, channels, format) as a hashref.

```
my $info = ainfo("song.mp3")
p $info->{duration}      # seconds
p $info->{sample_rate}   # e.g. 44100
```

id3_read

`id3_read` (alias `id3`) — read ID3 tags from an MP3 file as a hashref (title, artist, album, year, etc.).

```
my $tags = id3("song.mp3")
p "$tags->{artist} - $tags->{title}"
```

id3_write

`id3_write` (alias `id3w`) — write ID3 tags to an MP3 file from a hashref.

```
id3w("song.mp3", {title => "New Title", artist => "Artist", year => "2026"})
```

Network Utilities

18 topics

net_interfaces

net_interfaces (aliases net_ifs, ifconfig) — list all network interfaces with name, IPs, MAC, and status.

```
my @ifs = net_ifs()
for (@ifs) { p "_->{name}: _->{ipv4}" }
```

net_ipv4

net_ipv4 (aliases myip, myip4) — return the first non-loopback IPv4 address as a string.

```
p myip()   # e.g. 192.168.1.42
```

net_ipv6

net_ipv6 (alias myip6) — return the first non-loopback IPv6 address as a string.

```
p myip6()   # e.g. fe80::1
```

net_mac

net_mac (alias mymac) — return the first non-loopback MAC address.

```
p mymac()   # e.g. aa:bb:cc:dd:ee:ff
```

net_public_ip

net_public_ip (aliases pubip, extip) — fetch your public IP address via HTTP.

```
p pubip()   # e.g. 203.0.113.42
```

net_dns

net_dns (aliases dns_resolve, resolve) — resolve a hostname to IP addresses.

```
my @ips = resolve("github.com")
p @ips
```

net_reverse_dns

net_reverse_dns (alias rdns) — reverse DNS lookup via UDP PTR query.

```
p rdns("8.8.8.8")   # dns.google
```

net_ping

net_ping (alias ping) — TCP connect ping to host:port. Returns array of RTT measurements in ms.

```
my @rtt = ping("google.com", 443, 5)   # 5 pings
p mean(@rtt)
```

net_port_open

`net_port_open` (alias `port_open`) — check if a TCP port is open on a host. Returns boolean.

```
p port_open("localhost", 8080)   # 1 or 0
```

net_ports_scan

`net_ports_scan` (aliases `port_scan`, `portscan`) — scan a range of TCP ports on a host. Returns list of open ports.

```
my @open = port_scan("localhost", 80, 443)
p @open   # e.g. 80, 443
```

net_latency

`net_latency` (alias `tcplat`) — measure TCP connect latency to host:port in milliseconds.

```
p tcplat("google.com", 443)   # e.g. 12.5
```

net_download

`net_download` (aliases `download`, `wget`) — download a URL to a local file via ureq.

```
wget("https://example.com/data.csv", "data.csv")
```

net_headers

`net_headers` (alias `http_headers`) — fetch HTTP response headers as a hashref.

```
my $h = http_headers("https://example.com")
p $h->{"content-type"}
```

net_dns_servers

`net_dns_servers` (alias `dns_servers`) — return the system's configured DNS server addresses.

```
my @dns = dns_servers()
p @dns   # e.g. 8.8.8.8, 8.8.4.4
```

net_gateway

`net_gateway` (alias `gateway`) — return the default gateway IP address.

```
p gateway()   # e.g. 192.168.1.1
```

net_whois

`net_whois` (alias `whois`) — WHOIS lookup via raw TCP connection on port 43.

```
p whois("example.com")
```

net_hostname

`net_hostname` — return the system hostname.

```
p net_hostname()   # e.g. my-macbook.local
```

smtp_send

`smtp_send` (aliases `send_email`, `email`) — send an email via SMTP with TLS. Takes a hashref with to, from, subject, body, and SMTP connection fields.

```
smtp_send({to => "bob@ex.com", from => "me@ex.com", subject => "Hi",
  body => "Hello!", smtp_host => "smtp.ex.com", smtp_port => 587,
  smtp_user => "me@ex.com", smtp_pass => $pass})
```

Markup / Web Scraping

4 topics

html_parse

`html_parse` (alias `parse_html`) — parse an HTML string into an array of element hashrefs with tag, text, and attrs.

```
my @els = html_parse(slurp("page.html"))
for (@els) { p "_->{tag}: _->{text}" }
```

css_select

`css_select` (aliases `css`, `qs`, `query_selector`) — query parsed HTML with a CSS selector. Returns matching elements as hashrefs with tag, text, attrs, and html.

```
my @links = css_select(slurp("page.html"), "a.nav")
for (@links) { p "_->{tag} _->{attrs}{href}" }
```

xml_parse

`xml_parse` (alias `parse_xml`) — parse an XML string into a nested hashref tree with tag, text, attrs, and children.

```
my $root = xml_parse(slurp("data.xml"))
p $root->{tag}
for (@{$root->{children}}) { p _->{text} }
```

xpath

`xpath` (alias `xml_select`) — query XML with XPath-like expressions (//tag, //tag[@attr='val']). Returns matching nodes as hashrefs.

```
my @items = xpath(slurp("feed.xml"), "//item")
for (@items) { p _->{text} }
```

Date Utilities

4 topics

dateseq

`dateseq` (alias `dseq`) — generate a sequence of dates. Args: start, end [, step]. Step defaults to 1 day.

```
my @days = dseq("2026-01-01", "2026-01-07")
p @days
```

dategrep

`dategrep` (alias `dgrep`) — filter a list of date strings by a pattern or range.

```
my @jan = dgrep("2026-01", @dates)
```

dateround

`dateround` (alias `dround`) — round date strings to a unit (day, hour, month, etc.).

```
p dround("2026-04-19T14:35:22", "hour")   # 2026-04-19T15:00:00
```

datesort

`datesort` (alias `dsort`) — sort date strings chronologically.

```
my @sorted = dsort("2026-03-01", "2026-01-15", "2026-02-10")
p @sorted
```

Git

10 topics

git_log

`git_log` (alias `glog`) — get the last N commits as an array of hashrefs (hash, author, date, message).

```
my @commits = glog(10)
for (@commits) { p "_->{hash} _->{message}" }
```

git_status

`git_status` (alias `gst`) — get working tree status as an array of hashrefs (path, status).

```
my @st = gst()
for (@st) { p "_->{status} _->{path}" }
```

git_diff

`git_diff` (alias `gdiff`) — get the current diff as a string.

```
p gdiff()
gdiff() |> lines |> grep /^\+/ |> e p
```

git_branches

`git_branches` (alias `gbr`) — list all branches as an array of strings.

```
my @br = gbr()
p @br
```

git_tags

`git_tags` (alias `gtags`) — list all tags as an array of strings.

```
my @tags = gtags()
p @tags
```

git_blame

`git_blame` (alias `gblame`) — blame a file, returning per-line annotation.

```
p gblame("src/main.rs")
```

git_authors

`git_authors` (alias `gauthors`) — list unique authors sorted by commit count as hashrefs.

```
my @authors = gauthors()
for (@authors) { p "_->{name}: _->{count}" }
```

git_files

`git_files` (alias `gfiles`) — list all tracked files in the repo.

```
my @files = gfiles()
p scalar @files   # total tracked files
```

git_show

`git_show` (alias `gshow`) — show details of a commit (message, diff, author).

```
p gshow("HEAD")
p gshow("abc1234")
```

git_root

`git_root` (alias `groot`) — return the repository root path.

```
p groot()   # e.g. /home/user/project
```

System

13 topics

mounts

`mounts` (aliases `disk_mounts`, `filesystems`) — list mounted filesystems with usage (total, used, available).

```
my @m = mounts()
for (@m) { p "_->{mount}: _->{used}/_->{total}" }
```

thread_count

`thread_count` (alias `nthreads`) — return the rayon thread pool size.

```
p nthreads()   # e.g. 8
```

pool_info

`pool_info` (alias `par_info`) — return thread pool details as a hashref (threads, queued, active).

```
my $info = par_info()
p $info->{threads}
```

par_bench

`par_bench` (alias `pbench`) — run a parallel throughput benchmark and return results.

```
my $result = pbench(1000000)
p $result->{ops_per_sec}
```

to_pdf

`to_pdf` — generate a PDF from text, SVG, or structured data. Returns raw PDF bytes.

```
"Hello World" |> to_pdf |> to_file("out.pdf")
scatter_svg([1,2,3], [1,4,9]) |> to_pdf |> to_file("plot.pdf")
```

pdf_text

`pdf_text` (aliases `pdf_read`, `pdf_extract`) — extract all text content from a PDF file and return it as a string.

```
my $text = pdf_text("report.pdf")
p $text |> lines |> cnt   # number of lines
```

pdf_pages

`pdf_pages` — return the number of pages in a PDF file.

```
p pdf_pages("report.pdf")   # e.g. 42
```

jq

`jq` — alias for `json_jq`. Query JSON data with jq-style expressions.

```
my $data = json_decode(slurp("data.json"))
p jq($data, ".items[].name")
```

stress_cpu

`stress_cpu` (alias `scpu`) — CPU stress test: run SHA256 hashing in a tight loop for the specified duration (default 1 second). Returns number of hashes computed. Use for load testing and thermal validation.

```
my $hashes = stress_cpu(10)   # 10 seconds of CPU stress
p "Computed $hashes hashes"

# Distributed across cluster
my $c = cluster(["node1:8", "node2:8"])
1:16 |> pmap_on $c { stress_cpu(60) }   # 60s stress on all workers
```

stress_mem

`stress_mem` (alias `smem`) — Memory stress test: allocate and touch the specified number of bytes (default 100MB). Returns bytes allocated. Use for memory pressure testing and OOM validation.

```
my $bytes = stress_mem(1e9)   # Allocate 1GB
p "Allocated $bytes bytes"

# Memory pressure across cluster
1:16 |> pmap_on $cluster { stress_mem(4e9) }   # 4GB per worker
```

stress_io

`stress_io` (alias `sio`) — IO stress test: write and read temp files in the specified directory (default /tmp) for N iterations (default 100). Returns total bytes written. Use for storage IOPS testing.

```
my $bytes = stress_io("/tmp", 1000)   # 1000 iterations
p "Wrote $bytes bytes"

# IO stress across cluster
1:16 |> pmap_on $cluster { stress_io("/data", 500) }
```

stress_test

`stress_test` (alias `st`) — Combined CPU/memory/IO stress test. If cluster provided, distributes across all workers. Returns hashref with stats: {cpu_hashes, mem_bytes, io_bytes, workers, duration}.

```
# Local stress test (10 seconds)
my $r = stress_test(10)
p "CPU: $r->{cpu_hashes}, Workers: $r->{workers}"

# Distributed stress test
my $c = cluster(["node1:8", "node2:8", "node3:8"])
my $r = stress_test($c, 60)   # 60s across 24 workers
p "Hashes: $r->{cpu_hashes}, Duration: $r->{duration}s"
```

heat

`heat` — Maximum thermal stress. Pins ALL cores to 100% TDP. The hottest function in any programming language. Use for infrastructure burn-in, cooling validation, and power testing.

```
heat(60)     # 60 seconds of maximum heat
heat()       # default 60 seconds

# Output:
# □ HEAT: Pinning N cores to 100% TDP for 60s
# □ HEAT: 3,116,320,000 hashes in 60.00s (51.9M/s across N cores)
```

Testing

15 topics

assert_eq

`assert_eq` (alias `aeq`) — assert that two values are equal as strings. Takes A, B, and an optional message. Fails the test with a diff if A ne B.

```
assert_eq $got, $expected, "username matches"
aeq length(@arr), 3
```

assert_ne

`assert_ne` (alias `ane`) — assert that two values are not equal as strings. Takes A, B, and an optional message. Fails if A eq B.

```
assert_ne $token, "", "token must not be empty"
ane $a, $b
```

assert_ok

`assert_ok` (alias `aok`) — assert that a value is truthy (defined and non-zero/non-empty). Fails if the value is falsy or undef.

```
assert_ok $result, "fetch returned data"
aok $user->{active}
```

assert_err

`assert_err` — assert that a value is falsy or undef. The inverse of `assert_ok`. Useful for verifying error conditions or absent values.

```
assert_err $deleted_user, "user should be gone"
assert_err 0
```

assert_true

`assert_true` (alias `atrue`) — alias for `assert_ok`. Assert that a value is truthy.

```
assert_true $connected, "should be connected"
atrue $flag
```

assert_false

`assert_false` (alias `afalse`) — alias for `assert_err`. Assert that a value is falsy or undef.

```
assert_false $error, "no error expected"
afalse $disabled
```

assert_gt

`assert_gt` — assert that the first numeric value is strictly greater than the second. Fails with the actual values on mismatch.

```
assert_gt $elapsed, 0, "elapsed must be positive"
assert_gt scalar(@results), 10
```

assert_lt

`assert_lt` — assert that the first numeric value is strictly less than the second.

```
assert_lt $latency, 100, "latency under 100ms"
assert_lt $errors, $threshold
```

assert_ge

`assert_ge` — assert that the first numeric value is greater than or equal to the second.

```
assert_ge $count, 1, "at least one result"
assert_ge $version, 2
```

assert_le

`assert_le` — assert that the first numeric value is less than or equal to the second.

```
assert_le $memory, $limit, "within memory budget"
assert_le length($buf), 4096
```

assert_match

`assert_match` (alias `amatch`) — assert that a string matches a regex pattern. Fails with the actual string and pattern on mismatch.

```
assert_match $email, qr/\@/, "must contain @"
amatch $line, qr/^OK/
```

assert_contains

`assert_contains` (alias `acontains`) — assert that a string contains a given substring. Fails with both values on mismatch.

```
assert_contains $html, "<title>", "page has title"
acontains $log, "SUCCESS"
```

assert_near

`assert_near` (alias `anear`) — assert that two floats are approximately equal within an epsilon tolerance (default 1e-9). Essential for floating-point comparisons.

```
assert_near 0.1 + 0.2, 0.3, 1e-10, "float add"
anear $pi, 3.14159, 1e-5
```

assert_dies

`assert_dies` (alias `adies`) — assert that a block throws an error. Passes if the block dies, fails if it returns normally.

```
assert_dies { die "boom" } "should throw"
adies { 1 / 0 }
```

test_run

`test_run` (alias `run_tests`) — print a test summary with pass/fail counts and exit with code 1 if any test failed. Call at the end of a test file to report results.

```
# ... assertions above ...
test_run   # prints summary, exits 1 on failure
```

Test Runner (worker pool)

4 topics

test

`test(FILE_OR_DIR, ...)` — run every `test_*.stk` / `t_*.stk` under each path through the worker pool. Same engine as `s t` on the CLI: pre-forks N persistent stryke `--test-worker` processes, dispatches paths via JSON over stdin/stdout, each worker `fork()`s a grandchild per test for full isolation, no per-test dyld cost. Returns the exit code (0 on all-pass, 1 on any failure). Optional trailing hashref selects mode: `{ fork => 1 }` for the legacy `posix_spawn`-per-test path, `{ inproc => 1 }` for single-process VM-per-test (fastest, hermetic only), `{ quiet => 1 }`, `{ no_interop => 1 }`.

```
test("t/")                              # default: worker pool
test(["t/", "examples/"], { quiet => 1 })
test("t/test_foo.stk", { fork => 1 })   # opt back into per-test fork
my $rc = test(@dirs)                    # array flattens; rc = 0/1
```

test_no_interop

`test_no_interop(FILE_OR_DIR, ...)` (alias `test_ni`) — same as `test`, but pins each worker thread's TLS no-interop flag for the duration of the test (`set_no_interop_mode_tls`). Sibling `pmaps` workers don't race on a shared atomic. Forwards `--no-interop` to every spawned grandchild so each test runs under stryke's bot-firewall mode (rejects Perl-isms, forces stryke idioms).

```
test_no_interop("t/")
test_ni(["t/a", "t/b"], { quiet => 1 })
```

check

`check(FILE, ...)` — in-process equivalent of `stryke check FILE...`. Parses, compiles, and lints each path without executing it. Returns the exit code (0 on success, 1 on errors). Optional trailing hashref: `{ quiet => 1 }`, `{ json => 1 }`, `{ no_interop => 1 }`.

Faster than `system "stryke check $f"` because it skips the binary fork and reuses the parent process's parser/compiler. The lint pass is not a strict-vars enforcer — undefined-variable errors only fire on files that themselves do `use strict;`.

```
my $rc = check("foo.stk")                   # 0 = ok, 1 = errors
check(\@files, { quiet => 1 })              # arrayref + opts
for my $f (glob("src/*.stk")) { check($f) or warn "$f failed" }
```

check_no_interop

`check_no_interop(FILE, ...)` (alias `check_ni`) — same as `check`, but pins this thread to `--no-interop` for the duration of the check via an RAII guard on the per-thread no-interop flag. Sibling threads are unaffected, so the builtin is safe to call from `pmaps` workers in parallel — no global-atomic race.

```
check_no_interop("foo.stk")                 # rejects sub/say/$a/$b
pmaps { check_ni(_, { quiet => 1 }) } @files # parallel-safe, no race
```

AI primitives (docs/AI_PRIMITIVES.md)

79 topics

ai

`ai $prompt, opts...` — single-shot LLM call (or auto-routes to the agent loop when `tools => [...]`/`tool fn`s are in scope, vision when `image => $url|$path`, structured output when `schema => +{...}`, document input when `pdf => $path`).

```
my $r = ai "summarize this", $document
my $r = ai "weather in SF"   # auto-uses any `tool fn` defined in scope
my $r = ai "describe", image => "/photo.jpg"
my $r = ai "extract user", schema => +{ name => "string", age => "int" }
```

Providers: anthropic (default), openai. Mock with `ai_mock_install`. Cost via `ai_cost`.

stream_prompt

`stream_prompt $prompt, on_chunk => sub { ... }` — Anthropic SSE streaming. With `on_chunk`, fires once per text-delta. Without, returns a `PerlIterator`-backed handle: `for my $chunk in stream_prompt("...") { print $chunk }`.

```
my $state = +{ buf => "" };
stream_prompt("write a haiku",
    on_chunk => sub { $state->{buf} .= $_[0]; print $_[0] });
```

chat

`chat \@messages, opts...` — explicit message-list call. Each entry is `+{role => "user"|"assistant"|"system", content => ...}`.

```
chat([
    +{ role => "system", content => "Be concise" },
    +{ role => "user",   content => "hi" },
])
```

For multi-turn with auto-tracking, use `ai_session_*` instead.

embed

`embed $text` or `embed \@texts` — vector embedding via Voyage AI (default) or OpenAI. Returns arrayref of floats (single text) or arrayref-of-arrayref (batch). Routes via `ai_routing_set("embed", ...)`.

```
my @vec = @{ embed("hello world") }
my @vecs = @{ embed(["a", "b", "c"]) }
```

tokens_of

`tokens_of` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = tokens_of $x
# or in a pipeline:
@list |> map tokens_of |> p
```

ai_estimate

`ai_estimate` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_estimate $x
# or in a pipeline:
@list |> map ai_estimate |> p
```

ai_cost

`ai_cost()` — running spend / token / cache stats hashref: `+{usd, input_tokens, output_tokens, embed_tokens, cache_creation_tokens, cache_read_tokens, cache_hits, cache_misses}`.

ai_history

`ai_history` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_history $x
# or in a pipeline:
@list |> map ai_history |> p
```

ai_history_clear

`ai_history_clear` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_history_clear $x
# or in a pipeline:
@list |> map ai_history_clear |> p
```

ai_cache_clear

`ai_cache_clear` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_cache_clear $x
# or in a pipeline:
@list |> map ai_cache_clear |> p
```

ai_cache_size

`ai_cache_size` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_cache_size $x
# or in a pipeline:
@list |> map ai_cache_size |> p
```

ai_mock_install

`ai_mock_install` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_mock_install $x
# or in a pipeline:
@list |> map ai_mock_install |> p
```

ai_mock_clear

`ai_mock_clear` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_mock_clear $x
# or in a pipeline:
@list |> map ai_mock_clear |> p
```

ai_config_get

`ai_config_get` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_config_get $x
# or in a pipeline:
@list |> map ai_config_get |> p
```

ai_config_set

`ai_config_set` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_config_set $x
# or in a pipeline:
@list |> map ai_config_set |> p
```

ai_routing_get

`ai_routing_get` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_routing_get $x
# or in a pipeline:
@list |> map ai_routing_get |> p
```

ai_routing_set

`ai_routing_set` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_routing_set $x
# or in a pipeline:
@list |> map ai_routing_set |> p
```

ai_register_tool

`ai_register_tool($name, $description, +{params}, sub { ... })` — register a tool that the bare `ai($prompt)` call auto-attaches to its agent loop. Idempotent re-register on same name.

```
ai_register_tool("weather", "Get weather",
    +{ city => "string" },
    sub { fetch "https://api.weather.com/" . uri_encode($_[0]->{city}) });
ai_clear_tools();
```

For parser-level sugar, use `tool fn` instead.

ai_unregister_tool

`ai_unregister_tool` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_unregister_tool $x
# or in a pipeline:
@list |> map ai_unregister_tool |> p
```

ai_clear_tools

`ai_clear_tools` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_clear_tools $x
# or in a pipeline:
@list |> map ai_clear_tools |> p
```

ai_tools_list

`ai_tools_list` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_tools_list $x
# or in a pipeline:
@list |> map ai_tools_list |> p
```

ai_filter

`ai_filter \@items, "criterion"` — keep items the model judges as matching the criterion. Single batched LLM call; cheap by construction.

ai_map

`ai_map \@items, "instruction"` — apply instruction to each item, single batched LLM call. Returns arrayref of strings.

ai_classify

`ai_classify \@items, "hint", into => ["a", "b", "c"]` — assign one of the labels to each item.

ai_match

`ai_match` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_match $x
# or in a pipeline:
@list |> map ai_match |> p
```

ai_sort

`ai_sort` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_sort $x
# or in a pipeline:
@list |> map ai_sort |> p
```

ai_dedupe

`ai_dedupe` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_dedupe $x
# or in a pipeline:
@list |> map ai_dedupe |> p
```

ai_extract

`ai_extract "prompt", schema => +{ name => "string", age => "int" }` — structured JSON output; returns a typed hashref.

ai_summarize

`ai_summarize $text, words => 50` — convenience wrapper that asks the model for a concise summary at target length.

ai_translate

`ai_translate $text, to => "Spanish"` — convenience wrapper for translation.

ai_template

`ai_template "hi {name}", name => "world"` — pure string substitution. `{{` escapes to a literal `{`. Missing keys pass through unchanged.

ai_transcribe

`ai_transcribe "audio.mp3", model => "whisper-1", language => "en"` — OpenAI Whisper transcription. Multipart upload to `/v1/audio/transcriptions`. Returns the transcript string. Defaults to `whisper-1`.

ai_speak

`ai_speak $text, voice => "alloy", model => "tts-1", format => "mp3", output => "out.mp3"` — OpenAI TTS. Returns raw audio bytes; pass `output =>` to also write to disk. Voices: alloy, echo, fable, onyx, nova, shimmer.

ai_image

`ai_image $prompt, model => "dall-e-3", size => "1024x1024", quality => "hd", output => "out.png", n => 1` — image generation (OpenAI Images). Returns raw PNG bytes for n=1, or arrayref of bytes for n>1. Models: `dall-e-3`, `gpt-image-1`. `output` writes to disk (with `_1`/`_2`/... suffix for n>1).

ai_image_edit

`ai_image_edit $prompt, image => "in.png", mask => "mask.png", output => "out.png"` — image-to-image edit via OpenAI `/v1/images/edits`. Optional mask is a PNG with transparent regions marking where to edit. Returns PNG bytes (or arrayref for `n>1`).

ai_image_variation

`ai_image_variation image => "in.png", n => 4, size => "1024x1024"` — generate variations of an existing image (OpenAI `/v1/images/variations`, DALL-E 2). Returns PNG bytes for `n=1` or arrayref for `n>1`.

ai_models

`ai_models($provider)` — arrayref of model IDs from the live catalog. Providers: `openai`, `anthropic`, `ollama`, `gemini`. Useful for tooling that surfaces the available models without hardcoding lists.

ai_citations

`ai_citations()` — arrayref of citations from the most recent Anthropic call (when `citations => 1` was passed). Each entry is a hashref with `text` (cited string), `title`, `start`/`end` char offsets or `start_page`/`end_page` page numbers, and `document_index`.

ai_dashboard

`ai_dashboard()` — ANSI-colored multi-line summary (cost, tokens in/out, embed tokens, prompt-cache write/read, result-cache hit ratio). Drop at the end of a script or during a session: `print ai_dashboard()`.

ai_pricing

`ai_pricing("claude-opus-4-7")` — hashref `+{ model, input, output, input_per_1m, output_per_1m }` with the per-1K-token pricing the runtime uses for cost tracking. Useful for pre-flight cost estimates.

ai_describe

`ai_describe("image.png", style => "concise")` — vision wrapper that asks the model to describe an image. Styles: `concise` (one sentence), `detailed` (paragraph), `alt` (HTML alt-text ≤125 chars), or any custom suffix.

ai_grounded

`ai_grounded($prompt, documents => [@paths_or_strings], titles => [@titles])` — Anthropic-only multi-document grounding. Each entry can be a PDF path, a text-file path, or an inline string. Citations are auto-enabled; retrieve via `ai_citations()` after the call.

ai_moderate

`ai_moderate($text, model => "omni-moderation-latest")` → `+{ flagged, categories, scores }`. OpenAI content-safety classifier. Free endpoint — no token cost. Use it before sending user-generated content to a generative model.

ai_chunk

`ai_chunk($text, max_tokens => 500, overlap => 50, by => "tokens"|"chars"|"sentences")` → arrayref of strings. Pure local logic — no API call. Sentence mode splits on `.!?` and greedily packs.

ai_warm

`ai_warm(model => ..., provider => ...)` → `+{ ok, latency_ms, error, model, provider }`. Sends a 1-token ping so auth/network failures surface at script start instead of mid-flow.

ai_compare

`ai_compare($a, $b, criteria => "factual accuracy", scale => 5)` → `+{ winner, reason, scores => +{a, b}, raw }`. Single-call structured comparison. `winner` is `"a"`, `"b"`, or `"tie"`.

ai_session_export

`ai_session_export($handle)` → JSON string capturing system, model, provider, and full message log. Pair with `ai_session_import($json)` to restore.

ai_session_import

`ai_session_import($json)` → fresh session handle hashref. Inverse of `ai_session_export`. Allocates a new id and populates the session from the JSON body.

ai_file_upload

`ai_file_upload("path", purpose => "user_data")` → metadata hashref with `id`, `bytes`, `filename`, `purpose`, `created_at`. OpenAI Files API — file_id can be referenced by Whisper/Vision/Batch/Assistants.

ai_file_list

`ai_file_list(purpose => "...")` → arrayref of file metadata hashrefs from `/v1/files`.

ai_file_get

`ai_file_get($file_id)` → metadata hashref for a single file.

ai_file_delete

`ai_file_delete($file_id)` → 1 if deleted, 0 otherwise. Removes the file from OpenAI's storage.

ai_file_anthropic_upload

`ai_file_anthropic_upload("path/to/file.pdf")` → metadata hashref with `id`, `filename`, `mime_type`, `size_bytes`, `created_at`. Anthropic's beta `/v1/files` endpoint — file_id can be referenced from batch jobs and document blocks.

ai_file_anthropic_list

`ai_file_anthropic_list()` → arrayref of file-metadata hashrefs from Anthropic's `/v1/files`.

ai_file_anthropic_delete

`ai_file_anthropic_delete($file_id)` → 1 on success, 0 otherwise. Removes the file from Anthropic's storage.

ai_session_new

`ai_session_new(system => "...", model => "...")` — start a multi-turn chat session. Use `ai_session_send($s, "msg")` to advance, `ai_session_history($s)` to inspect, `ai_session_close($s)` to drop.

ai_session_send

`ai_session_send` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_session_send $x
# or in a pipeline:
@list |> map ai_session_send |> p
```

ai_session_history

`ai_session_history` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_session_history $x
# or in a pipeline:
@list |> map ai_session_history |> p
```

ai_session_close

`ai_session_close` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_session_close $x
# or in a pipeline:
@list |> map ai_session_close |> p
```

ai_session_reset

`ai_session_reset` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_session_reset $x
# or in a pipeline:
@list |> map ai_session_reset |> p
```

ai_memory_save

`ai_memory_save("id", "content", $metadata?, $path?)` — embed + persist a doc in the sqlite-backed memory store.

ai_memory_recall

`ai_memory_recall("query", top_k => 5)` — re-embed query, return top-k matches by cosine similarity. Each entry: `+{id, content, score, metadata}`.

ai_memory_forget

`ai_memory_forget` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_memory_forget $x
# or in a pipeline:
@list |> map ai_memory_forget |> p
```

ai_memory_count

`ai_memory_count` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_memory_count $x
# or in a pipeline:
@list |> map ai_memory_count |> p
```

ai_memory_clear

`ai_memory_clear` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_memory_clear $x
# or in a pipeline:
@list |> map ai_memory_clear |> p
```

ai_vision

`ai_vision` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_vision $x
# or in a pipeline:
@list |> map ai_vision |> p
```

ai_pdf

`ai_pdf` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_pdf $x
# or in a pipeline:
@list |> map ai_pdf |> p
```

ai_budget

`ai_budget($usd_cap, sub { ... })` — runs the block; raises a runtime error if AI spend during the block exceeds `$usd_cap`. Restores prior global cap on exit.

ai_last_thinking

`ai_last_thinking` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_last_thinking $x
# or in a pipeline:
@list |> map ai_last_thinking |> p
```

ai_batch

`ai_batch` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_batch $x
# or in a pipeline:
@list |> map ai_batch |> p
```

ai_pmap

`ai_pmap` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_pmap $x
# or in a pipeline:
@list |> map ai_pmap |> p
```

vec_cosine

`vec_cosine \@a, \@b` — cosine similarity in `[-1, 1]`. Both args are arrayrefs of floats (typical embedding shape).

vec_search

`vec_search \@query, \@candidates, top_k => N` — return top-k matches as `+{idx, score}`.

vec_topk

`vec_topk` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = vec_topk $x
# or in a pipeline:
@list |> map vec_topk |> p
```

web_search_tool

`web_search_tool()` — built-in tool spec for `tools => [...]`. Brave Search via `$BRAVE_SEARCH_API_KEY`, falls back to DuckDuckGo HTML scrape.

fetch_url_tool

`fetch_url_tool()` — built-in tool spec that fetches a URL and returns text.

read_file_tool

`read_file_tool()` — built-in tool spec that reads a file from disk.

run_code_tool

`run_code_tool()` — built-in tool spec that runs Python code in a subprocess (10s timeout, returns stdout+stderr).

tool

`tool fn NAME($p1: type, $p2: type) -> Type "docstring" { body }` — declare an agent-callable tool. Param types become the JSON Schema. Auto-registers via `ai_register_tool` so bare `ai($prompt)` calls see it.

```
tool fn weather($city: string) "Get current weather" {
    fetch "https://api.weather.com/" . uri_encode($city)
}

tool fn calc($a: int, $b: int) -> int "Add two ints" { $a + $b }

my $r = ai("what's the weather in SF and 7+5?");
```

MCP (Model Context Protocol)

13 topics

mcp_connect

`mcp_connect "stdio:CMD ARGS"` or `mcp_connect "https://server/mcp"` — connect to an MCP server. Speaks JSON-RPC over stdio (subprocess) or streamable HTTP. Returns a handle.

mcp_tools

`mcp_tools $h` — list the server's tools. Cached after first call.

mcp_resources

`mcp_resources` — mcp (model context protocol) builtin.

```
my $result = mcp_resources $x
# or in a pipeline:
@list |> map mcp_resources |> p
```

mcp_prompts

`mcp_prompts` — mcp (model context protocol) builtin.

```
my $result = mcp_prompts $x
# or in a pipeline:
@list |> map mcp_prompts |> p
```

mcp_call

`mcp_call $h, "tool_name", +{...args}` — invoke a tool on the connected server.

mcp_resource

`mcp_resource` — mcp (model context protocol) builtin.

```
my $result = mcp_resource $x
# or in a pipeline:
@list |> map mcp_resource |> p
```

mcp_prompt

`mcp_prompt` — mcp (model context protocol) builtin.

```
my $result = mcp_prompt $x
# or in a pipeline:
@list |> map mcp_prompt |> p
```

mcp_close

`mcp_close` — mcp (model context protocol) builtin.

```
my $result = mcp_close $x
# or in a pipeline:
@list |> map mcp_close |> p
```

mcp_attach_to_ai

`mcp_attach_to_ai $h` — register a connected MCP server so subsequent bare `ai($prompt)` calls auto-include its tools.

mcp_detach_from_ai

`mcp_detach_from_ai` — mcp (model context protocol) builtin.

```
my $result = mcp_detach_from_ai $x
# or in a pipeline:
@list |> map mcp_detach_from_ai |> p
```

mcp_attached

`mcp_attached` — mcp (model context protocol) builtin.

```
my $result = mcp_attached $x
# or in a pipeline:
@list |> map mcp_attached |> p
```

mcp_server_start

`mcp_server_start "name", +{ tools => [...] }` — run a stdio JSON-RPC MCP server in-process. Pair with `s_web build` to ship a self-contained MCP server binary.

For parser-level sugar, use `mcp_server "name" { tool foo($a) "..." {...} }`.

mcp_server

`mcp_server "name" { tool NAME(args) "doc" { body } ... }` — declarative MCP server block. Compiles to `mcp_server_start("name", +{ tools => [...] })` with private helper fns for each tool body.

```
mcp_server "filesystem" {
    tool read_file($path: string) "Read file contents" { slurp $path }
    tool list_dir($path: string)  "List dir entries"  { join("\n", readdir $path) }
}
```

Web framework (stryke_web/README.md)

112 topics

web_route

`web_route "VERB /path", "controller#action"` — register a route in the global router.

```
web_route "GET /posts", "posts#index"
web_route "POST /posts", "posts#create"
```

web_resources

`web_resources "posts"` — register the 7-route REST scaffold (index/show/new/create/edit/update/destroy) for a resource.

web_root

`web_root "posts#index"` — shortcut for `web_route "GET /"`.

web_application_config

`web_application_config` — web framework (docs/web_framework.md) builtin.

```
my $result = web_application_config $x
# or in a pipeline:
@list |> map web_application_config |> p
```

web_render

`web_render html => "..." / text => ... / json => ... / template => "posts/index", locals => +{...}` — emit a response. Templates resolve `app/views/...html.erb`, run through the ERB engine, wrap in the application layout.

web_render_partial

`web_render_partial` — web framework (docs/web_framework.md) builtin.

```
my $result = web_render_partial $x
# or in a pipeline:
@list |> map web_render_partial |> p
```

web_redirect

`web_redirect "/path", 302` — issue a 3xx redirect with `Location` header.

web_json

`web_json` — web framework (docs/web_framework.md) builtin.

```
my $result = web_json $x
# or in a pipeline:
@list |> map web_json |> p
```

web_text

`web_text` — web framework (docs/web_framework.md) builtin.

```
my $result = web_text $x
# or in a pipeline:
@list |> map web_text |> p
```

web_params

`web_params()` — hashref of query + form-urlencoded body + JSON body + path captures.

web_request

`web_request` — web framework (docs/web_framework.md) builtin.

```
my $result = web_request $x
# or in a pipeline:
@list |> map web_request |> p
```

web_routes_table

`web_routes_table` — web framework (docs/web_framework.md) builtin.

```
my $result = web_routes_table $x
# or in a pipeline:
@list |> map web_routes_table |> p
```

web_boot_application

`web_boot_application` — web framework (docs/web_framework.md) builtin.

```
my $result = web_boot_application $x
# or in a pipeline:
@list |> map web_boot_application |> p
```

web_db_connect

`web_db_connect "sqlite://path"` — open and cache the global SQLite connection. Required before any `web_model_*` call.

web_db_execute

`web_db_execute` — web framework (docs/web_framework.md) builtin.

```
my $result = web_db_execute $x
# or in a pipeline:
@list |> map web_db_execute |> p
```

web_db_query

`web_db_query` — web framework (docs/web_framework.md) builtin.

```
my $result = web_db_query $x
# or in a pipeline:
@list |> map web_db_query |> p
```

web_jobs_init

`web_jobs_init()` — create the SQLite `jobs` table if it doesn't exist. Idempotent.

web_job_enqueue

`web_job_enqueue("name", +{...args}, queue => "...", priority => N, max_attempts => N)` → job id (i64). Status starts as `pending`.

web_job_dequeue

`web_job_dequeue(queue => "...")` → `+{id, name, args_json, args, attempts, max_attempts}` or undef. Atomically marks the picked job as `running` and increments `attempts`.

web_job_complete

`web_job_complete($id)` — mark a running job as `done`.

web_job_fail

`web_job_fail($id, error => "msg")` — if `attempts < max_attempts`, returns to `pending` for retry; otherwise marked `failed`. Returns the new status.

web_jobs_list

`web_jobs_list(queue => "...", status => "...", limit => 50)` → arrayref of job hashrefs.

web_jobs_stats

`web_jobs_stats()` → `+{pending, running, done, failed}`.

web_job_purge

`web_job_purge(status => "done")` → number of rows deleted.

web_db_begin

`web_db_begin` — web framework (docs/web_framework.md) builtin.

```
my $result = web_db_begin $x
# or in a pipeline:
@list |> map web_db_begin |> p
```

web_db_commit

`web_db_commit` — web framework (docs/web_framework.md) builtin.

```
my $result = web_db_commit $x
# or in a pipeline:
@list |> map web_db_commit |> p
```

web_db_rollback

`web_db_rollback` — web framework (docs/web_framework.md) builtin.

```
my $result = web_db_rollback $x
# or in a pipeline:
@list |> map web_db_rollback |> p
```

web_model_all

`web_model_all "posts"` — `SELECT * FROM posts ORDER BY id`. Returns arrayref of hashrefs.

web_model_find

`web_model_find "posts", $id` — find by id; returns hashref or undef.

web_model_where

`web_model_where` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_where $x
# or in a pipeline:
@list |> map web_model_where |> p
```

web_model_create

`web_model_create "posts", +{title => "...", body => "..."}` — INSERT + return new row with id and auto-set `created_at`/`updated_at`.

web_model_update

`web_model_update` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_update $x
# or in a pipeline:
@list |> map web_model_update |> p
```

web_model_destroy

`web_model_destroy` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_destroy $x
# or in a pipeline:
@list |> map web_model_destroy |> p
```

web_model_soft_destroy

`web_model_soft_destroy` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_soft_destroy $x
# or in a pipeline:
@list |> map web_model_soft_destroy |> p
```

web_model_paginate

`web_model_paginate` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_paginate $x
# or in a pipeline:
@list |> map web_model_paginate |> p
```

web_model_search

`web_model_search` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_search $x
# or in a pipeline:
@list |> map web_model_search |> p
```

web_model_count

`web_model_count` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_count $x
# or in a pipeline:
@list |> map web_model_count |> p
```

web_model_first

`web_model_first` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_first $x
# or in a pipeline:
@list |> map web_model_first |> p
```

web_model_last

`web_model_last` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_last $x
# or in a pipeline:
@list |> map web_model_last |> p
```

web_model_increment

`web_model_increment` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_increment $x
# or in a pipeline:
@list |> map web_model_increment |> p
```

web_model_with

`web_model_with` — web framework (docs/web_framework.md) builtin.

```
my $result = web_model_with $x
# or in a pipeline:
@list |> map web_model_with |> p
```

web_create_table

`web_create_table` — web framework (docs/web_framework.md) builtin.

```
my $result = web_create_table $x
# or in a pipeline:
@list |> map web_create_table |> p
```

web_drop_table

`web_drop_table` — web framework (docs/web_framework.md) builtin.

```
my $result = web_drop_table $x
# or in a pipeline:
@list |> map web_drop_table |> p
```

web_add_column

`web_add_column` — web framework (docs/web_framework.md) builtin.

```
my $result = web_add_column $x
# or in a pipeline:
@list |> map web_add_column |> p
```

web_remove_column

web_remove_column — web framework (docs/web_framework.md) builtin.

```
my $result = web_remove_column $x
# or in a pipeline:
@list |> map web_remove_column |> p
```

web_migrate

web_migrate() — apply every up block from db/migrate/*.stk whose version isn't yet in schema_migrations.

web_rollback

web_rollback — web framework (docs/web_framework.md) builtin.

```
my $result = web_rollback $x
# or in a pipeline:
@list |> map web_rollback |> p
```

web_validate

web_validate $attrs, +{title => "presence,length:1..100", email => "format:^.+@.+$"} — returns +{ok => 1} or +{ok => 0, errors => +{...}}. Validators: presence, length:MIN..MAX, format:REGEX, numericality, inclusion:a|b|c, confirmation:other_field.

web_permit

web_permit — web framework (docs/web_framework.md) builtin.

```
my $result = web_permit $x
# or in a pipeline:
@list |> map web_permit |> p
```

web_password_hash

web_password_hash $pw returns a web1$saltHex$digest shaped string. web_password_verify $pw, $stored returns 1/0. SHA-256 + 16-byte salt; constant-time compare.

web_session

web_session — web framework (docs/web_framework.md) builtin.

```
my $result = web_session $x
# or in a pipeline:
@list |> map web_session |> p
```

web_session_set

Server-side session storage backed by a signed cookie. web_session_set("user_id", 42) / web_session_get("user_id"). Cookie value is base64-JSON; HttpOnly + SameSite=Lax by default.

web_session_clear

`web_session_clear` — web framework (docs/web_framework.md) builtin.

```
my $result = web_session_clear $x
# or in a pipeline:
@list |> map web_session_clear |> p
```

web_set_cookie

`web_set_cookie` — web framework (docs/web_framework.md) builtin.

```
my $result = web_set_cookie $x
# or in a pipeline:
@list |> map web_set_cookie |> p
```

web_cookies

`web_cookies` — web framework (docs/web_framework.md) builtin.

```
my $result = web_cookies $x
# or in a pipeline:
@list |> map web_cookies |> p
```

web_flash

`web_flash` — web framework (docs/web_framework.md) builtin.

```
my $result = web_flash $x
# or in a pipeline:
@list |> map web_flash |> p
```

web_flash_set

`web_flash_set` — web framework (docs/web_framework.md) builtin.

```
my $result = web_flash_set $x
# or in a pipeline:
@list |> map web_flash_set |> p
```

web_flash_get

`web_flash_get` — web framework (docs/web_framework.md) builtin.

```
my $result = web_flash_get $x
# or in a pipeline:
@list |> map web_flash_get |> p
```

web_jwt_encode

`web_jwt_encode +{user_id => 1, role => "admin"}` — HS256-sign a JWT using `secret_key_base` from `web_application_config`.

web_jwt_decode

`web_jwt_decode $token` — verify + decode an HS256 JWT. Returns the payload hashref or undef on bad signature.

web_bearer_token

`web_bearer_token` — web framework (docs/web_framework.md) builtin.

```
my $result = web_bearer_token $x
# or in a pipeline:
@list |> map web_bearer_token |> p
```

web_otp_secret

TOTP / 2FA — RFC 6238, 30-second window, ±1 step skew tolerance. `web_otp_secret()` returns a base32 secret to share with the authenticator app; `web_otp_generate($secret)` returns the current 6-digit code; `web_otp_verify($secret, $code)` returns 1/0.

web_token_for

`web_token_for` — web framework (docs/web_framework.md) builtin.

```
my $result = web_token_for $x
# or in a pipeline:
@list |> map web_token_for |> p
```

web_token_consume

`web_token_consume` — web framework (docs/web_framework.md) builtin.

```
my $result = web_token_consume $x
# or in a pipeline:
@list |> map web_token_consume |> p
```

web_can

`web_can` — web framework (docs/web_framework.md) builtin.

```
my $result = web_can $x
# or in a pipeline:
@list |> map web_can |> p
```

web_signed

`web_signed` — web framework (docs/web_framework.md) builtin.

```
my $result = web_signed $x
# or in a pipeline:
@list |> map web_signed |> p
```

web_unsigned

`web_unsigned` — web framework (docs/web_framework.md) builtin.

```
my $result = web_unsigned $x
# or in a pipeline:
@list |> map web_unsigned |> p
```

web_cache_get

`web_cache_get` — web framework (docs/web_framework.md) builtin.

```
my $result = web_cache_get $x
# or in a pipeline:
@list |> map web_cache_get |> p
```

web_cache_set

`web_cache_set` — web framework (docs/web_framework.md) builtin.

```
my $result = web_cache_set $x
# or in a pipeline:
@list |> map web_cache_set |> p
```

web_cache_delete

`web_cache_delete` — web framework (docs/web_framework.md) builtin.

```
my $result = web_cache_delete $x
# or in a pipeline:
@list |> map web_cache_delete |> p
```

web_cache_clear

`web_cache_clear` — web framework (docs/web_framework.md) builtin.

```
my $result = web_cache_clear $x
# or in a pipeline:
@list |> map web_cache_clear |> p
```

web_t

`web_t` — web framework (docs/web_framework.md) builtin.

```
my $result = web_t $x
# or in a pipeline:
@list |> map web_t |> p
```

web_load_locale

`web_load_locale` — web framework (docs/web_framework.md) builtin.

```
my $result = web_load_locale $x
# or in a pipeline:
@list |> map web_load_locale |> p
```

web_log

`web_log` — web framework (docs/web_framework.md) builtin.

```
my $result = web_log $x
# or in a pipeline:
@list |> map web_log |> p
```

web_set_header

`web_set_header` — web framework (docs/web_framework.md) builtin.

```
my $result = web_set_header $x
# or in a pipeline:
@list |> map web_set_header |> p
```

web_status

`web_status` — web framework (docs/web_framework.md) builtin.

```
my $result = web_status $x
# or in a pipeline:
@list |> map web_status |> p
```

web_uuid

web_uuid — web framework (docs/web_framework.md) builtin.

```
my $result = web_uuid $x
# or in a pipeline:
@list |> map web_uuid |> p
```

web_now

web_now — web framework (docs/web_framework.md) builtin.

```
my $result = web_now $x
# or in a pipeline:
@list |> map web_now |> p
```

web_security_headers

web_security_headers — web framework (docs/web_framework.md) builtin.

```
my $result = web_security_headers $x
# or in a pipeline:
@list |> map web_security_headers |> p
```

web_openapi

web_openapi — web framework (docs/web_framework.md) builtin.

```
my $result = web_openapi $x
# or in a pipeline:
@list |> map web_openapi |> p
```

web_etag

web_etag — web framework (docs/web_framework.md) builtin.

```
my $result = web_etag $x
# or in a pipeline:
@list |> map web_etag |> p
```

web_csv

web_csv — web framework (docs/web_framework.md) builtin.

```
my $result = web_csv $x
# or in a pipeline:
@list |> map web_csv |> p
```

web_markdown

web_markdown — web framework (docs/web_framework.md) builtin.

```
my $result = web_markdown $x
# or in a pipeline:
@list |> map web_markdown |> p
```

web_md

`web_md` — web framework (docs/web_framework.md) builtin. Alias for `web_markdown`.

```
my $result = web_md $x
# or in a pipeline:
@list |> map web_md |> p
```

web_content_for

`web_content_for` — web framework (docs/web_framework.md) builtin.

```
my $result = web_content_for $x
# or in a pipeline:
@list |> map web_content_for |> p
```

web_yield_content

`web_yield_content` — web framework (docs/web_framework.md) builtin.

```
my $result = web_yield_content $x
# or in a pipeline:
@list |> map web_yield_content |> p
```

web_jsonapi_resource

`web_jsonapi_resource` — web framework (docs/web_framework.md) builtin.

```
my $result = web_jsonapi_resource $x
# or in a pipeline:
@list |> map web_jsonapi_resource |> p
```

web_jsonapi_collection

`web_jsonapi_collection` — web framework (docs/web_framework.md) builtin.

```
my $result = web_jsonapi_collection $x
# or in a pipeline:
@list |> map web_jsonapi_collection |> p
```

web_jsonapi_error

`web_jsonapi_error` — web framework (docs/web_framework.md) builtin.

```
my $result = web_jsonapi_error $x
# or in a pipeline:
@list |> map web_jsonapi_error |> p
```

web_rate_limit

`web_rate_limit` — web framework (docs/web_framework.md) builtin.

```
my $result = web_rate_limit $x
# or in a pipeline:
@list |> map web_rate_limit |> p
```

web_before_action

`web_before_action` — web framework (docs/web_framework.md) builtin.

```
my $result = web_before_action $x
# or in a pipeline:
@list |> map web_before_action |> p
```

web_after_action

web_after_action — web framework (docs/web_framework.md) builtin.

```
my $result = web_after_action $x
# or in a pipeline:
@list |> map web_after_action |> p
```

web_h

web_h — web framework (docs/web_framework.md) builtin.

```
my $result = web_h $x
# or in a pipeline:
@list |> map web_h |> p
```

web_link_to

web_link_to — web framework (docs/web_framework.md) builtin.

```
my $result = web_link_to $x
# or in a pipeline:
@list |> map web_link_to |> p
```

web_button_to

web_button_to — web framework (docs/web_framework.md) builtin.

```
my $result = web_button_to $x
# or in a pipeline:
@list |> map web_button_to |> p
```

web_form_with

web_form_with — web framework (docs/web_framework.md) builtin.

```
my $result = web_form_with $x
# or in a pipeline:
@list |> map web_form_with |> p
```

web_form_close

web_form_close — web framework (docs/web_framework.md) builtin.

```
my $result = web_form_close $x
# or in a pipeline:
@list |> map web_form_close |> p
```

web_text_field

web_text_field — web framework (docs/web_framework.md) builtin.

```
my $result = web_text_field $x
# or in a pipeline:
@list |> map web_text_field |> p
```

web_text_area

`web_text_area` — web framework (docs/web_framework.md) builtin.

```
my $result = web_text_area $x
# or in a pipeline:
@list |> map web_text_area |> p
```

web_check_box

`web_check_box` — web framework (docs/web_framework.md) builtin.

```
my $result = web_check_box $x
# or in a pipeline:
@list |> map web_check_box |> p
```

web_csrf_meta_tag

`web_csrf_meta_tag` — web framework (docs/web_framework.md) builtin.

```
my $result = web_csrf_meta_tag $x
# or in a pipeline:
@list |> map web_csrf_meta_tag |> p
```

web_stylesheet_link_tag

`web_stylesheet_link_tag` — web framework (docs/web_framework.md) builtin.

```
my $result = web_stylesheet_link_tag $x
# or in a pipeline:
@list |> map web_stylesheet_link_tag |> p
```

web_javascript_link_tag

`web_javascript_link_tag` — web framework (docs/web_framework.md) builtin.

```
my $result = web_javascript_link_tag $x
# or in a pipeline:
@list |> map web_javascript_link_tag |> p
```

web_image_tag

`web_image_tag` — web framework (docs/web_framework.md) builtin.

```
my $result = web_image_tag $x
# or in a pipeline:
@list |> map web_image_tag |> p
```

web_truncate

`web_truncate` — web framework (docs/web_framework.md) builtin.

```
my $result = web_truncate $x
# or in a pipeline:
@list |> map web_truncate |> p
```

web_pluralize

`web_pluralize` — web framework (docs/web_framework.md) builtin.

```
my $result = web_pluralize $x
# or in a pipeline:
@list |> map web_pluralize |> p
```

web_time_ago_in_words

`web_time_ago_in_words` — web framework (docs/web_framework.md) builtin.

```
my $result = web_time_ago_in_words $x
# or in a pipeline:
@list |> map web_time_ago_in_words |> p
```

web_faker_name

`web_faker_name` — web framework (docs/web_framework.md) builtin.

```
my $result = web_faker_name $x
# or in a pipeline:
@list |> map web_faker_name |> p
```

web_faker_email

`web_faker_email` — web framework (docs/web_framework.md) builtin.

```
my $result = web_faker_email $x
# or in a pipeline:
@list |> map web_faker_email |> p
```

web_faker_sentence

`web_faker_sentence` — web framework (docs/web_framework.md) builtin.

```
my $result = web_faker_sentence $x
# or in a pipeline:
@list |> map web_faker_sentence |> p
```

web_faker_paragraph

`web_faker_paragraph` — web framework (docs/web_framework.md) builtin.

```
my $result = web_faker_paragraph $x
# or in a pipeline:
@list |> map web_faker_paragraph |> p
```

web_faker_int

`web_faker_int` — web framework (docs/web_framework.md) builtin.

```
my $result = web_faker_int $x
# or in a pipeline:
@list |> map web_faker_int |> p
```

Stress / telemetry (docs/STRESS_TESTING.md)

42 topics

stress_fp

`stress_fp $secs` — float matrix-multiply pinning every core. Returns total FLOP count.

stress_int

`stress_int $secs` — integer pipeline pinning every core.

stress_cache

`stress_cache $secs, $kb` — pound a working set of `kb` KiB per core in random order. Default 1024 (~ L2/L3 boundary).

stress_branch

`stress_branch` — stress / telemetry extensions builtin.

```
my $result = stress_branch $x
# or in a pipeline:
@list |> map stress_branch |> p
```

stress_sort

`stress_sort` — stress / telemetry extensions builtin.

```
my $result = stress_sort $x
# or in a pipeline:
@list |> map stress_sort |> p
```

stress_alloc

`stress_alloc` — stress / telemetry extensions builtin.

```
my $result = stress_alloc $x
# or in a pipeline:
@list |> map stress_alloc |> p
```

stress_mmap

`stress_mmap` — stress / telemetry extensions builtin.

```
my $result = stress_mmap $x
# or in a pipeline:
@list |> map stress_mmap |> p
```

stress_disk

`stress_disk $path, $secs, $mb_per_write` — sustained sequential write + fsync + read across all cores.

stress_iops

`stress_iops $path, $secs, $block_kb` — small random reads/writes for IOPS testing. Default 4KB blocks.

stress_net

`stress_net "host:port", $secs, $conns` — open `conns` TCP connections per core, pump bytes.

stress_http

`stress_http $url, $secs, $conns` — HTTP GET storm. Counts 2xx responses.

stress_dns

`stress_dns` — stress / telemetry extensions builtin.

```
my $result = stress_dns $x
# or in a pipeline:
@list |> map stress_dns |> p
```

stress_thread

`stress_thread` — stress / telemetry extensions builtin.

```
my $result = stress_thread $x
# or in a pipeline:
@list |> map stress_thread |> p
```

stress_fork

`stress_fork` — stress / telemetry extensions builtin.

```
my $result = stress_fork $x
# or in a pipeline:
@list |> map stress_fork |> p
```

stress_aes

`stress_aes $secs` — AES-128 encrypt round-trip across cores.

stress_compress

`stress_compress $secs, $kb` — gzip compress+decompress payload of `kb` KiB per core.

stress_regex

`stress_regex` — stress / telemetry extensions builtin.

```
my $result = stress_regex $x
# or in a pipeline:
@list |> map stress_regex |> p
```

stress_json

`stress_json` — stress / telemetry extensions builtin.

```
my $result = stress_json $x
# or in a pipeline:
@list |> map stress_json |> p
```

stress_burst

`stress_burst` — stress / telemetry extensions builtin.

```
my $result = stress_burst $x
# or in a pipeline:
@list |> map stress_burst |> p
```

stress_ramp

`stress_ramp` — stress / telemetry extensions builtin.

```
my $result = stress_ramp $x
# or in a pipeline:
@list |> map stress_ramp |> p
```

stress_oscillate

`stress_oscillate` — stress / telemetry extensions builtin.

```
my $result = stress_oscillate $x
# or in a pipeline:
@list |> map stress_oscillate |> p
```

stress_all

`stress_all $secs` — run every stress kernel above in parallel for `secs`. Returns a hashref of per-kernel counts.

stress_cores

`stress_cores` — stress / telemetry extensions builtin.

```
my $result = stress_cores $x
# or in a pipeline:
@list |> map stress_cores |> p
```

stress_temp

Telemetry helpers: `stress_temp()` reads CPU temp from `/sys/class/thermal` (Linux), `stress_freq()` averages cpufreq across cores, `stress_throttled()` returns 1 when freq < 80% of max AND temp > 80°C, `stress_load()` returns `+{m1, m5, m15}`, `stress_meminfo()` returns total/used/free/swap bytes.

stress_thermal_zones

`stress_thermal_zones` — stress / telemetry extensions builtin.

```
my $result = stress_thermal_zones $x
# or in a pipeline:
@list |> map stress_thermal_zones |> p
```

stress_arm_kill_switch

`stress_arm_kill_switch` — stress / telemetry extensions builtin.

```
my $result = stress_arm_kill_switch $x
# or in a pipeline:
@list |> map stress_arm_kill_switch |> p
```

stress_killed

`stress_killed` — stress / telemetry extensions builtin.

```
my $result = stress_killed $x
# or in a pipeline:
@list |> map stress_killed |> p
```

stress_disarm_kill_switch

`stress_disarm_kill_switch` — stress / telemetry extensions builtin.

```
my $result = stress_disarm_kill_switch $x
# or in a pipeline:
@list |> map stress_disarm_kill_switch |> p
```

stress_metrics_record

`stress_metrics_record("name", $value, k1 => v1, ...)` — append a sample to the metrics history. Labels are stringified.

stress_metrics_clear

`stress_metrics_clear()` — wipe the metrics history. Pair with `stress_metrics_count()` before/after to verify.

stress_metrics_count

`stress_metrics_count()` → current number of samples in the history.

stress_metrics_export

`stress_metrics_export("path", format => "csv"|"json"|"prom")` → number of samples written.

stress_metrics_prometheus

`stress_metrics_prometheus()` → string in Prometheus text exposition format. Latest value per (name, labels) wins. Drop into a `/metrics` HTTP handler.

stress_metrics_json

`stress_metrics_json()` → JSON string of the entire metrics history.

stress_metrics_csv

`stress_metrics_csv()` → CSV string with `ts_ms,name,value,labels` columns.

stress_metrics_watch

`stress_metrics_watch(field => "stress_temp", interval_ms => 1000, max_ticks => 60)` → arrayref of values sampled at the requested cadence. Polls the metrics history for new entries matching `field`.

audit_log

`audit_log("event_name", k1 => v1, ...)` — append one JSONL line to `~/.stryke/audit.log` (or `$STRYKE_AUDIT_LOG`). Never throws; returns 1/0.

audit_log_path

`audit_log_path()` → absolute path the next `audit_log` call will write to.

secrets_encrypt

`secrets_encrypt($plain, key => $key)` → base64 envelope (nonce(12) || ciphertext || tag(16)). AES-256-GCM. Key must be 32 raw bytes or 32-byte base64.

secrets_decrypt

`secrets_decrypt($envelope, key => $key)` → plaintext, or undef on auth failure.

secrets_random_key

`secrets_random_key()` → fresh 32-byte AES-256 key as base64. Reads `/dev/urandom` on Unix.

secrets_kdf

`secrets_kdf($password, salt => "...", iterations => 600_000)` → 32-byte key as base64. PBKDF2-HMAC-SHA256.

PTY / expect (docs/expect-feature-idea.md)

13 topics

pty_spawn

`pty_spawn "cmd args"` or `pty_spawn("cmd", "arg1", "arg2")` — allocate a PTY, fork+exec the child, return a handle. Master fd is non-blocking.

```
my $h = pty_spawn("ssh user@host");
pty_expect($h, qr/password:/, 30);
pty_send($h, "hunter2\n");
pty_close($h);
```

pty_send

`pty_send $h, "text"` — write to the PTY master. Handles EAGAIN/EINTR; returns bytes written.

pty_read

`pty_read` — mcp (model context protocol) builtin.

```
my $result = pty_read $x
# or in a pipeline:
@list |> map pty_read |> p
```

pty_expect

`pty_expect $h, qr/regex/, $timeout_secs` — loop reading until the regex matches the accumulated buffer or `timeout_secs` elapses. Returns the matched substring or undef on timeout/EOF. Buffer is consumed up to (and including) the match.

pty_expect_table

`pty_expect_table $h, [+{re => qr/.../, do => sub {...}}, ...], $timeout` — multi-pattern dispatch (Tcl `expect { ... }` block). First match wins; the matched branch's `do` sub is called.

pty_buffer

`pty_buffer` — mcp (model context protocol) builtin.

```
my $result = pty_buffer $x
# or in a pipeline:
@list |> map pty_buffer |> p
```

pty_alive

`pty_alive` — mcp (model context protocol) builtin.

```
my $result = pty_alive $x
# or in a pipeline:
@list |> map pty_alive |> p
```

pty_eof

`pty_eof` — mcp (model context protocol) builtin.

```
my $result = pty_eof $x
# or in a pipeline:
@list |> map pty_eof |> p
```

pty_close

`pty_close $h` — SIGTERM → 200ms grace → SIGKILL. Returns exit status.

pty_interact

`pty_interact $h` — raw-mode handoff: forwards stdin → master and master → stdout until EOF or Ctrl-]. Saves and restores tty state.

pty_strip_ansi

`pty_strip_ansi $text` → string with CSI/OSC/ESC sequences removed. Useful before regex-matching against PTY output that mixes color codes into prompts.

pty_after_eof

`pty_after_eof $h, "callback_name"` — register an EOF callback. Spawns a watcher thread that flips a fired flag once the PTY closes. Pair with `pty_pending_events()` in your main loop.

pty_pending_events

`pty_pending_events()` → arrayref of `+{handle_id, callback}` for any `pty_after_eof` registrations that have fired since the last call. Drains the fired list.

Other

6716 topics

%pc

parsec (alias pc) — pc ≈ 3.086×10^16 m ≈ 3.26 ly. Common astronomical distance unit.

```
p pc        # 3.0856775814913673e16 m
p pc / ly   # ~3.26 light years
```

%stryke::aliases

%stryke::aliases (short: %a) — alias spelling → canonical primary.

Keys are the 2nd-and-later names in each try_builtin match arm. For O(1) *reverse* lookup (primary → all its aliases), use %stryke::primaries / %p.

```
p $a{tj}                              # "to_json"
p $a{bn}                              # "basename"
p scalar keys %a                      # total alias count
```

%stryke::all

%stryke::all (short: %all) — every callable *spelling* (primaries and aliases) → category. Aliases inherit their primary's category.

Use %all when you want "how many names can I type?" or want to look up an alias's category without hopping through %aliases. Use %builtins when you want unique operations.

```
p scalar keys %all   # total callable-spellings count
p $all{tj}           # "serialization"  (alias resolves via inheritance)
p $all{to_json}      # "serialization"
keys %all |> pager   # browse every spelling
```

%stryke::builtins

%stryke::builtins (short: %b) — every primary callable name → its category. Primaries-only, so scalar keys %b is a clean unique-operation count. For the "everything you can type" view (primaries + aliases), use %stryke::all / %all.

```
p $b{pmap}              # "parallel"
p $b{to_json}           # "serialization"
p scalar keys %b        # unique-op count
```

%stryke::categories

%stryke::categories (short: %c) — category string → arrayref of names in that category.

Inverted index on %builtins. Gives O(1) reverse-lookup for "list every op of kind X" queries that would otherwise be O(n) greps. Name lists are alphabetized.

```
$c{parallel} |> e p                 # every parallel op
p scalar @{ $c{parallel} }          # how many?
p join ", ", @{ $c{"array / list"} } # joined roster
keys %c |> sort |> p                # all category names
```

%stryke::descriptions

`%stryke::descriptions` (short: `%d`) — name → one-line summary.

First sentence of each LSP hover doc (`doc_for_label_text`), harvested at build time. Sparse — only names that have a hover doc appear, so `exists $d{$name}` doubles as "is this documented?".

```
p $d{pmap}                              # one-line summary
p $d{to_json}                           # "Serialize a PerlValue to a JSON string."
p scalar keys %d                        # count of documented ops
keys %d |> grep { $d{_} =~ /parallel/i } |> sort |> p
```

%stryke::extensions

`%stryke::extensions` (short: `%e`) — stryke-only names, name → category.

Disjoint from `%perl_compats`. Everything `--compat` mode rejects at parse time, plus dispatch primaries like `basename`/`ddump` that are extensions at runtime even without a parser entry.

```
p $e{pmap}                              # "parallel"
keys %e |> grep /^p/ |> sort |> p       # every p* parallel op
```

%stryke::perl_compats

`%stryke::perl_compats` (short: `%pc`) — Perl 5 core names only, name → category.

Subset of `%builtins` restricted to names from `is_perl5_core`. Direct O(1) access for the "show me just Perl core" query.

```
p $pc{map}                  # "array / list"
p scalar keys %pc           # core-only count
keys %pc |> sort |> p       # enumerate every Perl core name
```

%stryke::primaries

`%stryke::primaries` (short: `%p`) — primary dispatcher name → arrayref of its aliases.

Inverted `%aliases`. Primaries with no aliases still have an entry (empty arrayref), so `exists $p{foo}` reliably answers "is foo a dispatch primary?" O(1).

```
$p{to_json} |> e p              # ["tj"]
p scalar @{ $p{basename} }      # how many aliases does basename have?
# find every primary that has at least one alias:
keys %p |> grep { scalar @{$p{_}} } |> sort |> p
```

CORE

`CORE::` — pseudo-namespace alias for every dispatch primary. Every callable bare name is also reachable as `CORE::name`, with identical semantics. The qualified spelling exists for tab-complete affordance and explicit-namespace style; stryke rejects redefining a builtin (`fn print { ... }` is a parse error), so there is nothing to shadow at runtime.

The set is built dynamically: any name in `keys %stryke::builtins` is a valid `CORE::name`. There's no separate `%CORE::` table — it's a virtual namespace resolved at dispatch time.

```
CORE::print "hello world\n"                 # □ print "hello world\n"
my $r = CORE::sum(1, 2, 3)                  # 6 — qualified call
map { CORE::sprintf("%d", _) } @nums        # any builtin
```

The `s docs CORE::name` form resolves to the bare-name page (the alias and primary share docs). Aliases (`tj`, `bn`, `e`, ...) also support `CORE::tj` / `CORE::bn` / `CORE::e`.

CORE::pipe

Create a unidirectional pipe, returning a pair of connected filehandles: one for reading and one for writing. Data written to the write end can be read from the read end, making pipes the fundamental building block for inter-process communication. Commonly used

with `fork` so the parent and child can exchange data.

```
pipe(my $rd, my $wr) or die "pipe: $!"
if (fork() == 0) {
    close($rd)
    print $wr "hello from child\n"
    exit(0)
}
close($wr)
my $msg = <$rd>
p $msg   # hello from child
```

a0

`bohr_radius` (alias `a0`) — a_0 ≈ 5.29×10^{-11} m. Radius of hydrogen atom ground state.

```
p a0   # 5.29177210903e-11
```

a256

`a256` — color operations builtin. Alias for `ansi_256`.

```
my $result = a256 $x
# or in a pipeline:
@list |> map a256 |> p
```

a3c_advantage_step

`a3c_advantage_step` — batch 40 builtin.

```
my $result = a3c_advantage_step $x
# or in a pipeline:
@list |> map a3c_advantage_step |> p
```

a85d

`base85_decode` (aliases `b85d`, `a85d`) decodes an Ascii85/Base85 encoded string.

```
p b85d(b85e("Hello"))  # Hello
```

a85e

`base85_encode` (aliases `b85e`, `a85e`) encodes a string using Ascii85/Base85 encoding. More compact than Base64 (4:5 ratio vs 3:4).

```
p b85e("Hello")  # encoded string
```

a_law_decode

`a_law_decode` — batch 25 builtin.

```
my $result = a_law_decode $x
# or in a pipeline:
@list |> map a_law_decode |> p
```

a_law_encode

`a_law_encode` — batch 25 builtin.

```
my $result = a_law_encode $x
# or in a pipeline:
@list |> map a_law_encode |> p
```

ab1_step

`ab1_step` — batch 34 builtin.

```
my $result = ab1_step $x
# or in a pipeline:
@list |> map ab1_step |> p
```

ab2_step

`ab2_step` — batch 34 builtin.

```
my $result = ab2_step $x
# or in a pipeline:
@list |> map ab2_step |> p
```

ab3_step

`ab3_step` — batch 34 builtin.

```
my $result = ab3_step $x
# or in a pipeline:
@list |> map ab3_step |> p
```

ab4

`ab4` — batch 19 builtin. Alias for `adams_bashforth_4`.

```
my $result = ab4 $x
# or in a pipeline:
@list |> map ab4 |> p
```

ab_coeff_sum

`ab_coeff_sum` — batch 34 builtin.

```
my $result = ab_coeff_sum $x
# or in a pipeline:
@list |> map ab_coeff_sum |> p
```

abelianization_quotient

`abelianization_quotient` — batch 37 builtin.

```
my $result = abelianization_quotient $x
# or in a pipeline:
@list |> map abelianization_quotient |> p
```

aberration_annual

`aberration_annual` — batch 55 builtin.

```
my $result = aberration_annual $x
# or in a pipeline:
@list |> map aberration_annual |> p
```

aberth_step

`aberth_step` — batch 19 builtin.

```
my $result = aberth_step $x
# or in a pipeline:
@list |> map aberth_step |> p
```

abs_ceil

`abs_ceil` — trivial numeric helpers (batch 4) builtin.

```
my $result = abs_ceil $x
# or in a pipeline:
@list |> map abs_ceil |> p
```

abs_diff

`abs_diff` — trivial numeric / predicate builtins builtin.

```
my $result = abs_diff $x
# or in a pipeline:
@list |> map abs_diff |> p
```

abs_each

`abs_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = abs_each $x
# or in a pipeline:
@list |> map abs_each |> p
```

abs_floor

`abs_floor` — trivial numeric helpers (batch 4) builtin.

```
my $result = abs_floor $x
# or in a pipeline:
@list |> map abs_floor |> p
```

absolute_magnitude

`absolute_magnitude` — batch 30 builtin.

```
my $result = absolute_magnitude $x
# or in a pipeline:
@list |> map absolute_magnitude |> p
```

absolute_vorticity_step

`absolute_vorticity_step` — batch 42 builtin.

```
my $result = absolute_vorticity_step $x
# or in a pipeline:
@list |> map absolute_vorticity_step |> p
```

absorbed_short_wave

`absorbed_short_wave` — batch 42 builtin.

```
my $result = absorbed_short_wave $x
# or in a pipeline:
@list |> map absorbed_short_wave |> p
```

abundant_numbers

`abundant_numbers` — number theory / primes builtin.

```
my $result = abundant_numbers $x
# or in a pipeline:
@list |> map abundant_numbers |> p
```

abunnums

`abunnums` — number theory / primes builtin. Alias for `abundant_numbers`.

```
my $result = abunnums $x
# or in a pipeline:
@list |> map abunnums |> p
```

abuse_test_voltage

`abuse_test_voltage` — batch 38 builtin.

```
my $result = abuse_test_voltage $x
# or in a pipeline:
@list |> map abuse_test_voltage |> p
```

accept_lang_pick

`accept_lang_pick` — batch 81 builtin.

```
my $result = accept_lang_pick $x
# or in a pipeline:
@list |> map accept_lang_pick |> p
```

acceptance_target

`acceptance_target` — batch 19 builtin.

```
my $result = acceptance_target $x
# or in a pipeline:
@list |> map acceptance_target |> p
```

accrint

`accrint` — batch 81 builtin.

```
my $result = accrint $x
# or in a pipeline:
@list |> map accrint |> p
```

accrued_interest

`accrued_interest` — batch 57 builtin.

```
my $result = accrued_interest $x
# or in a pipeline:
@list |> map accrued_interest |> p
```

accum

`accum` — python/ruby stdlib builtin. Alias for `accumulate`.

```
my $result = accum $x
# or in a pipeline:
@list |> map accum |> p
```

accumulate

`accumulate` — python/ruby stdlib builtin.

```
my $result = accumulate $x
# or in a pipeline:
@list |> map accumulate |> p
```

acf

`acf_fn` (alias `acf`) — autocorrelation function. Returns ACF values for lags 0..max_lag. Like R's acf().

```
my @a = @{acf([1,3,2,4,3,5,4,6], 5)}
p $a[0]  # 1.0 (lag 0 is always 1)
p $a[1]  # lag-1 autocorrelation
```

acf_basis

`acf_basis` — batch 78 builtin.

```
my $result = acf_basis $x
# or in a pipeline:
@list |> map acf_basis |> p
```

acirc

`acirc` — geometry / physics builtin. Alias for `area_circle`.

```
my $result = acirc $x
# or in a pipeline:
@list |> map acirc |> p
```

acorr

`acorr` — math / numeric (uncategorized batch) builtin. Alias for `autocorrelation`.

```
my $result = acorr $x
# or in a pipeline:
@list |> map acorr |> p
```

acos

`acos` — trig / math (batch 2) builtin.

```
my $result = acos $x
# or in a pipeline:
@list |> map acos |> p
```

acosh

`acosh` — trig / math (batch 2) builtin.

```
my $result = acosh $x
# or in a pipeline:
@list |> map acosh |> p
```

acot

`acot` — trig extensions builtin.

```
my $result = acot $x
# or in a pipeline:
@list |> map acot |> p
```

acro

`acro` — string processing (uncategorized batch) builtin. Alias for `acronym`.

```
my $result = acro $x
# or in a pipeline:
@list |> map acro |> p
```

acronym

`acronym` — string processing (uncategorized batch) builtin.

```
my $result = acronym $x
# or in a pipeline:
@list |> map acronym |> p
```

acsc

`acsc` — trig extensions builtin.

```
my $result = acsc $x
# or in a pipeline:
@list |> map acsc |> p
```

adaboost_alpha

`adaboost_alpha` — batch 27 builtin.

```
my $result = adaboost_alpha $x
# or in a pipeline:
@list |> map adaboost_alpha |> p
```

adagrad_step

`adagrad_step` — batch 19 builtin.

```
my $result = adagrad_step $x
# or in a pipeline:
@list |> map adagrad_step |> p
```

adams_bashforth_4

`adams_bashforth_4` — batch 19 builtin.

```
my $result = adams_bashforth_4 $x
# or in a pipeline:
@list |> map adams_bashforth_4 |> p
```

add_days

`add_days` — extended stdlib builtin.

```
my $result = add_days $x
# or in a pipeline:
@list |> map add_days |> p
```

add_hours

`add_hours` — extended stdlib builtin.

```
my $result = add_hours $x
# or in a pipeline:
@list |> map add_hours |> p
```

add_minutes

`add_minutes` — extended stdlib builtin.

```
my $result = add_minutes $x
# or in a pipeline:
@list |> map add_minutes |> p
```

addd

`addd` — extended stdlib builtin. Alias for `add_days`.

```
my $result = addd $x
# or in a pipeline:
@list |> map addd |> p
```

addh

`addh` — extended stdlib builtin. Alias for `add_hours`.

```
my $result = addh $x
# or in a pipeline:
@list |> map addh |> p
```

addm

`addm` — extended stdlib builtin. Alias for `add_minutes`.

```
my $result = addm $x
# or in a pipeline:
@list |> map addm |> p
```

adf_test_stat

`adf_test_stat` — batch 36 builtin.

```
my $result = adf_test_stat $x
# or in a pipeline:
@list |> map adf_test_stat |> p
```

adfuller

`adfuller` — batch 78 builtin.

```
my $result = adfuller $x
# or in a pipeline:
@list |> map adfuller |> p
```

adiabatic_lapse_rate_dry

`adiabatic_lapse_rate_dry` — batch 42 builtin.

```
my $result = adiabatic_lapse_rate_dry $x
# or in a pipeline:
@list |> map adiabatic_lapse_rate_dry |> p
```

adiabatic_lapse_rate_moist

`adiabatic_lapse_rate_moist` — batch 42 builtin.

```
my $result = adiabatic_lapse_rate_moist $x
# or in a pipeline:
@list |> map adiabatic_lapse_rate_moist |> p
```

adj_r2

`adj_r2` — batch 9 builtin. Alias for `adjusted_r_squared`.

```
my $result = adj_r2 $x
# or in a pipeline:
@list |> map adj_r2 |> p
```

adjacent_difference

`adjacent_difference` — list helpers (batch 4) builtin.

```
my $result = adjacent_difference $x
# or in a pipeline:
@list |> map adjacent_difference |> p
```

adjacent_pairs

`adjacent_pairs` — go/general functional utilities builtin.

```
my $result = adjacent_pairs $x
# or in a pipeline:
@list |> map adjacent_pairs |> p
```

adjp

`adjp` — go/general functional utilities builtin. Alias for `adjacent_pairs`.

```
my $result = adjp $x
# or in a pipeline:
@list |> map adjp |> p
```

adjusted_r_squared

`adjusted_r_squared` — batch 9 builtin.

```
my $result = adjusted_r_squared $x
# or in a pipeline:
@list |> map adjusted_r_squared |> p
```

adjusted_winner_pct

`adjusted_winner_pct` — batch 43 builtin.

```
my $result = adjusted_winner_pct $x
# or in a pipeline:
@list |> map adjusted_winner_pct |> p
```

adl32

`adl32` — extended stdlib builtin. Alias for `adler32`.

```
my $result = adl32 $x
# or in a pipeline:
@list |> map adl32 |> p
```

adler32

`adler32` — extended stdlib builtin.

```
my $result = adler32 $x
# or in a pipeline:
@list |> map adler32 |> p
```

adler32_hash

`adler32_hash` — batch 11 builtin.

```
my $result = adler32_hash $x
# or in a pipeline:
@list |> map adler32_hash |> p
```

adm_mass_step

`adm_mass_step` — batch 39 builtin.

```
my $result = adm_mass_step $x
# or in a pipeline:
@list |> map adm_mass_step |> p
```

admm_dual_step

`admm_dual_step` — batch 41 builtin.

```
my $result = admm_dual_step $x
# or in a pipeline:
@list |> map admm_dual_step |> p
```

admm_primal_step

`admm_primal_step` — batch 41 builtin.

```
my $result = admm_primal_step $x
# or in a pipeline:
@list |> map admm_primal_step |> p
```

ads_metric_step

`ads_metric_step` — batch 39 builtin.

```
my $result = ads_metric_step $x
# or in a pipeline:
@list |> map ads_metric_step |> p
```

aellip

`aellip` — geometry / physics builtin. Alias for `area_ellipse`.

```
my $result = aellip $x
# or in a pipeline:
@list |> map aellip |> p
```

aerosol_optical_depth

`aerosol_optical_depth` — batch 42 builtin.

```
my $result = aerosol_optical_depth $x
# or in a pipeline:
@list |> map aerosol_optical_depth |> p
```

aes_inv_sbox_byte

`aes_inv_sbox_byte` — batches 14-16 builtin.

```
my $result = aes_inv_sbox_byte $x
# or in a pipeline:
@list |> map aes_inv_sbox_byte |> p
```

aes_sbox_byte

`aes_sbox_byte` — batches 14-16 builtin.

```
my $result = aes_sbox_byte $x
# or in a pipeline:
@list |> map aes_sbox_byte |> p
```

affine_encrypt

`affine_encrypt` — batch 26 builtin.

```
my $result = affine_encrypt $x
# or in a pipeline:
@list |> map affine_encrypt |> p
```

african_monsoon_index

`african_monsoon_index` — batch 42 builtin.

```
my $result = african_monsoon_index $x
# or in a pipeline:
@list |> map african_monsoon_index |> p
```

after_n

`after_n` — functional combinators builtin.

```
my $result = after_n $x
# or in a pipeline:
@list |> map after_n |> p
```

age_at_date

`age_at_date` — batch 82 builtin.

```
my $result = age_at_date $x
# or in a pipeline:
@list |> map age_at_date |> p
```

age_at_z

`age_at_z` — batch 31 builtin.

```
my $result = age_at_z $x
# or in a pipeline:
@list |> map age_at_z |> p
```

age_from_birthdate

`age_from_birthdate` — batch 13 builtin.

```
my $result = age_from_birthdate $x
# or in a pipeline:
@list |> map age_from_birthdate |> p
```

age_in_years

`age_in_years` — extended stdlib batch 3 builtin.

```
my $result = age_in_years $x
# or in a pipeline:
@list |> map age_in_years |> p
```

ageyrs

`ageyrs` — extended stdlib batch 3 builtin. Alias for `age_in_years`.

```
my $result = ageyrs $x
# or in a pipeline:
@list |> map ageyrs |> p
```

ai_agent

`ai_agent` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = ai_agent $x
# or in a pipeline:
@list |> map ai_agent |> p
```

aic

`aic` — batch 9 builtin.

```
my $result = aic $x
# or in a pipeline:
@list |> map aic |> p
```

aig_simplify_step

`aig_simplify_step` — batch 67 builtin.

```
my $result = aig_simplify_step $x
# or in a pipeline:
@list |> map aig_simplify_step |> p
```

ain

`ain` — algebraic match builtin. Alias for `assoc_in`.

```
my $result = ain $x
# or in a pipeline:
@list |> map ain |> p
```

air_mass_kasten

`air_mass_kasten` — batches 14-16 builtin.

```
my $result = air_mass_kasten $x
# or in a pipeline:
@list |> map air_mass_kasten |> p
```

airy_disk_radius

`airy_disk_radius` — batch 55 builtin.

```
my $result = airy_disk_radius $x
# or in a pipeline:
@list |> map airy_disk_radius |> p
```

airyai

`airyai` — batch 72 builtin.

```
my $result = airyai $x
# or in a pipeline:
@list |> map airyai |> p
```

airybi

`airybi` — batch 72 builtin.

```
my $result = airybi $x
# or in a pipeline:
@list |> map airybi |> p
```

aitken_delta_squared

`aitken_delta_squared` — batches 14-16 builtin.

```
my $result = aitken_delta_squared $x
# or in a pipeline:
@list |> map aitken_delta_squared |> p
```

akaike_info_crit

`akaike_info_crit` — batch 36 builtin.

```
my $result = akaike_info_crit $x
# or in a pipeline:
@list |> map akaike_info_crit |> p
```

aks_witness_count

`aks_witness_count` — batch 35 builtin.

```
my $result = aks_witness_count $x
# or in a pipeline:
@list |> map aks_witness_count |> p
```

albedo_blackbody_balance

`albedo_blackbody_balance` — batch 42 builtin.

```
my $result = albedo_blackbody_balance $x
# or in a pipeline:
@list |> map albedo_blackbody_balance |> p
```

albers_conic_project

`albers_conic_project` — batches 14-16 builtin.

```
my $result = albers_conic_project $x
# or in a pipeline:
@list |> map albers_conic_project |> p
```

alexander_polynomial_at_one

`alexander_polynomial_at_one` — batch 37 builtin.

```
my $result = alexander_polynomial_at_one $x
# or in a pipeline:
@list |> map alexander_polynomial_at_one |> p
```

alfven_speed

`alfven_speed` — batch 23 builtin.

```
my $result = alfven_speed $x
# or in a pipeline:
@list |> map alfven_speed |> p
```

algebraic_intersection_number

`algebraic_intersection_number` — batch 37 builtin.

```
my $result = algebraic_intersection_number $x
# or in a pipeline:
@list |> map algebraic_intersection_number |> p
```

aliquot_sum

`aliquot_sum` — math / numeric (uncategorized batch) builtin.

```
my $result = aliquot_sum $x
# or in a pipeline:
@list |> map aliquot_sum |> p
```

all_distinct

`all_distinct` — more list helpers builtin.

```
my $result = all_distinct $x
# or in a pipeline:
@list |> map all_distinct |> p
```

all_eq

`all_eq` — more list helpers builtin.

```
my $result = all_eq $x
# or in a pipeline:
@list |> map all_eq |> p
```

all_match

`all_match` — predicates (batch 4) builtin.

```
my $result = all_match $x
# or in a pipeline:
@list |> map all_match |> p
```

all_pay_auction_step

`all_pay_auction_step` — batch 43 builtin.

```
my $result = all_pay_auction_step $x
# or in a pipeline:
@list |> map all_pay_auction_step |> p
```

all_unique

`all_unique` — more list helpers builtin. Alias for `all_distinct`.

```
my $result = all_unique $x
# or in a pipeline:
@list |> map all_unique |> p
```

allee_growth_step

`allee_growth_step` — batch 22 builtin.

```
my $result = allee_growth_step $x
# or in a pipeline:
@list |> map allee_growth_step |> p
```

allele_frequency

`allele_frequency` — batch 33 builtin.

```
my $result = allele_frequency $x
# or in a pipeline:
@list |> map allele_frequency |> p
```

alpha_fs

`fine_structure_constant` (alias `alpha_fs`) — $\alpha \approx 1/137 \approx 0.00730$. Fundamental constant of electromagnetism.

```
p alpha_fs       # 0.0072973525693
p 1 / alpha_fs   # ~137.036
```

alt_az_to_ra_dec

`alt_az_to_ra_dec` — batch 55 builtin.

```
my $result = alt_az_to_ra_dec $input
```

alternate_case

`alternate_case` — string helpers (batch 4) builtin.

```
my $result = alternate_case $x
# or in a pipeline:
@list |> map alternate_case |> p
```

alternating_seq_sum

`alternating_seq_sum` — batches 14-16 builtin.

```
my $result = alternating_seq_sum $x
# or in a pipeline:
@list |> map alternating_seq_sum |> p
```

always_false

`always_false` — functional primitives builtin.

```
my $result = always_false $x
# or in a pipeline:
@list |> map always_false |> p
```

always_true

`always_true` — functional primitives builtin.

```
my $result = always_true $x
# or in a pipeline:
@list |> map always_true |> p
```

am2_step

`am2_step` — batch 34 builtin.

```
my $result = am2_step $x
# or in a pipeline:
@list |> map am2_step |> p
```

am3_step

`am3_step` — batch 34 builtin.

```
my $result = am3_step $x
# or in a pipeline:
@list |> map am3_step |> p
```

am4_step

`am4_step` — batch 34 builtin.

```
my $result = am4_step $x
# or in a pipeline:
@list |> map am4_step |> p
```

amari_alpha_div

`amari_alpha_div` — batch 40 builtin.

```
my $result = amari_alpha_div $x
# or in a pipeline:
@list |> map amari_alpha_div |> p
```

amax

`amax` — extended stdlib builtin. Alias for `argmax`.

```
my $result = amax $x
# or in a pipeline:
@list |> map amax |> p
```

amin

`amin` — extended stdlib builtin. Alias for `argmin`.

```
my $result = amin $x
# or in a pipeline:
@list |> map amin |> p
```

amo_index_step

amo_index_step — batch 42 builtin.

```
my $result = amo_index_step $x
# or in a pipeline:
@list |> map amo_index_step |> p
```

amort

amortization_schedule($principal, $rate, $periods) (alias amort) — generates a full amortization schedule. Returns arrayref of hashrefs with period, payment, principal, interest, balance.

```
my $sched = amort(10000, 0.05, 12)
for my $row (@$sched) {
    p "Period $row->{period}: bal=$row->{balance}"
}
```

amortization_total_interest

amortization_total_interest — batch 13 builtin.

```
my $result = amortization_total_interest $x
# or in a pipeline:
@list |> map amortization_total_interest |> p
```

amp_to_db

amp_to_db — batch 30 builtin.

```
my $result = amp_to_db $input
```

amphoteric_check

amphoteric_check — batch 63 builtin.

```
my $result = amphoteric_check $x
# or in a pipeline:
@list |> map amphoteric_check |> p
```

amplitude_damping_channel

amplitude_damping_channel — batches 14-16 builtin.

```
my $result = amplitude_damping_channel $x
# or in a pipeline:
@list |> map amplitude_damping_channel |> p
```

amplitude_damping_excited

amplitude_damping_excited — batch 32 builtin.

```
my $result = amplitude_damping_excited $x
# or in a pipeline:
@list |> map amplitude_damping_excited |> p
```

amu

amu — physics constants builtin. Alias for atomic_mass_unit.

```
my $result = amu $x
# or in a pipeline:
@list |> map amu |> p
```

anagram_q

`anagram_q` — batch 11 builtin.

```
my $result = anagram_q $x
# or in a pipeline:
@list |> map anagram_q |> p
```

anaphora_distance

`anaphora_distance` — batch 69 builtin.

```
my $result = anaphora_distance $x
# or in a pipeline:
@list |> map anaphora_distance |> p
```

ancilla_alloc

`ancilla_alloc` — batch 80 builtin.

```
my $result = ancilla_alloc $x
# or in a pipeline:
@list |> map ancilla_alloc |> p
```

and

`and` — control flow builtin.

```
my $result = and $x
# or in a pipeline:
@list |> map and |> p
```

and_list

`and_list` — additional missing stdlib functions builtin.

```
my $result = and_list $x
# or in a pipeline:
@list |> map and_list |> p
```

anderson_darling

`anderson_darling` — batch 53 builtin.

```
my $result = anderson_darling $x
# or in a pipeline:
@list |> map anderson_darling |> p
```

anderson_step

`anderson_step` — batch 19 builtin.

```
my $result = anderson_step $x
# or in a pipeline:
@list |> map anderson_step |> p
```

andl

`andl` — additional missing stdlib functions builtin. Alias for `and_list`.

```
my $result = andl $x
# or in a pipeline:
@list |> map andl |> p
```

andoyer_dist

`andoyer_dist` — batch 58 builtin.

```
my $result = andoyer_dist $x
# or in a pipeline:
@list |> map andoyer_dist |> p
```

andrew_monotone_chain

`andrew_monotone_chain` — batches 17-18 builtin.

```
my $result = andrew_monotone_chain $x
# or in a pipeline:
@list |> map andrew_monotone_chain |> p
```

angbet

`angbet` — geometry (extended) builtin. Alias for `angle_between`.

```
my $result = angbet $x
# or in a pipeline:
@list |> map angbet |> p
```

angle_between

`angle_between($x1, $y1, $x2, $y2)` — computes the angle in radians between two 2D vectors from origin. Returns the angle using atan2.

```
my $angle = angle_between(1, 0, 0, 1)
p $angle   # ~1.5708 (π/2 radians = 90°)
```

angle_between_deg

`angle_between_deg` — math functions builtin.

```
my $result = angle_between_deg $x
# or in a pipeline:
@list |> map angle_between_deg |> p
```

angle_bracket

`angle_bracket` — string helpers (batch 4) builtin.

```
my $result = angle_bracket $x
# or in a pipeline:
@list |> map angle_bracket |> p
```

angular_diameter_distance

`angular_diameter_distance` — batch 31 builtin.

```
my $result = angular_diameter_distance $x
# or in a pipeline:
@list |> map angular_diameter_distance |> p
```

angular_velocity

`angular_velocity` — physics formulas builtin.

```
my $result = angular_velocity $x
# or in a pipeline:
@list |> map angular_velocity |> p
```

angvel

`angvel` — physics formulas builtin. Alias for `angular_velocity`.

```
my $result = angvel $x
# or in a pipeline:
@list |> map angvel |> p
```

annuity_due_an

`annuity_due_an` — batch 60 builtin.

```
my $result = annuity_due_an $x
# or in a pipeline:
@list |> map annuity_due_an |> p
```

annuity_future_value

`annuity_future_value` — batch 13 builtin.

```
my $result = annuity_future_value $x
# or in a pipeline:
@list |> map annuity_future_value |> p
```

annuity_immediate_an

`annuity_immediate_an` — batch 60 builtin.

```
my $result = annuity_immediate_an $x
# or in a pipeline:
@list |> map annuity_immediate_an |> p
```

annuity_present_value

`annuity_present_value` — batch 13 builtin.

```
my $result = annuity_present_value $x
# or in a pipeline:
@list |> map annuity_present_value |> p
```

anova

`anova_oneway` (aliases `anova`, `anova1`) performs one-way ANOVA. Returns [F-statistic, df_between, df_within].

```
my ($F, $dfb, $dfw) = @{anova([1,2,3], [4,5,6], [7,8,9])}
```

ansi_256

ansi_256 — color operations builtin.

```
my $result = ansi_256 $x
# or in a pipeline:
@list |> map ansi_256 |> p
```

ansi_black

ansi_black — color / ansi builtin.

```
my $result = ansi_black $x
# or in a pipeline:
@list |> map ansi_black |> p
```

ansi_blue

ansi_blue — color / ansi builtin.

```
my $result = ansi_blue $x
# or in a pipeline:
@list |> map ansi_blue |> p
```

ansi_bold

ansi_bold — color / ansi builtin.

```
my $result = ansi_bold $x
# or in a pipeline:
@list |> map ansi_bold |> p
```

ansi_cyan

ansi_cyan — color / ansi builtin.

```
my $result = ansi_cyan $x
# or in a pipeline:
@list |> map ansi_cyan |> p
```

ansi_dim

ansi_dim — color / ansi builtin.

```
my $result = ansi_dim $x
# or in a pipeline:
@list |> map ansi_dim |> p
```

ansi_green

ansi_green — color / ansi builtin.

```
my $result = ansi_green $x
# or in a pipeline:
@list |> map ansi_green |> p
```

ansi_magenta

ansi_magenta — color / ansi builtin.

```
my $result = ansi_magenta $x
# or in a pipeline:
@list |> map ansi_magenta |> p
```

ansi_off

`ansi_off` — color / ansi builtin. Alias for `red`.

```
my $result = ansi_off $x
# or in a pipeline:
@list |> map ansi_off |> p
```

ansi_red

`ansi_red` — color / ansi builtin.

```
my $result = ansi_red $x
# or in a pipeline:
@list |> map ansi_red |> p
```

ansi_reverse

`ansi_reverse` — color / ansi builtin.

```
my $result = ansi_reverse $x
# or in a pipeline:
@list |> map ansi_reverse |> p
```

ansi_truecolor

`ansi_truecolor` — color operations builtin.

```
my $result = ansi_truecolor $x
# or in a pipeline:
@list |> map ansi_truecolor |> p
```

ansi_underline

`ansi_underline` — color / ansi builtin.

```
my $result = ansi_underline $x
# or in a pipeline:
@list |> map ansi_underline |> p
```

ansi_white

`ansi_white` — color / ansi builtin.

```
my $result = ansi_white $x
# or in a pipeline:
@list |> map ansi_white |> p
```

ansi_yellow

`ansi_yellow` — color / ansi builtin.

```
my $result = ansi_yellow $x
# or in a pipeline:
@list |> map ansi_yellow |> p
```

anti_de_sitter_radius_step

anti_de_sitter_radius_step — batch 39 builtin.

```
my $result = anti_de_sitter_radius_step $x
# or in a pipeline:
@list |> map anti_de_sitter_radius_step |> p
```

antoine_p

antoine_p — batch 21 builtin.

```
my $result = antoine_p $x
# or in a pipeline:
@list |> map antoine_p |> p
```

any_match

any_match — predicates (batch 4) builtin.

```
my $result = any_match $x
# or in a pipeline:
@list |> map any_match |> p
```

aov

aov — batch 53 builtin.

```
my $result = aov $x
# or in a pipeline:
@list |> map aov |> p
```

ap_jonker_volgenant_step

ap_jonker_volgenant_step — batch 41 builtin.

```
my $result = ap_jonker_volgenant_step $x
# or in a pipeline:
@list |> map ap_jonker_volgenant_step |> p
```

apery

apery — math constants builtin. Alias for apery_constant.

```
my $result = apery $x
# or in a pipeline:
@list |> map apery |> p
```

apery_constant

apery_constant — math constants builtin.

```
my $result = apery_constant $x
# or in a pipeline:
@list |> map apery_constant |> p
```

apparent_horizon_step

apparent_horizon_step — batch 39 builtin.

```
my $result = apparent_horizon_step $x
# or in a pipeline:
@list |> map apparent_horizon_step |> p
```

append_elem

`append_elem` — list helpers (batch 4) builtin.

```
my $result = append_elem $x
# or in a pipeline:
@list |> map append_elem |> p
```

append_redis

`append_redis` — batch 71 builtin.

```
my $result = append_redis $x
# or in a pipeline:
@list |> map append_redis |> p
```

appl

`appl` — algebraic match builtin. Alias for `apply`.

```
my $result = appl $x
# or in a pipeline:
@list |> map appl |> p
```

apply

`apply` — algebraic match builtin.

```
my $result = apply $x
# or in a pipeline:
@list |> map apply |> p
```

apply_list

`apply_list` — functional combinators builtin.

```
my $result = apply_list $x
# or in a pipeline:
@list |> map apply_list |> p
```

apply_window

`apply_window(\@signal, \@window)` — element-wise multiplies signal by window function. Use before FFT to reduce spectral leakage.

```
my $w = hann(scalar @signal)
my $windowed = apply_window(\@signal, $w)
my $spectrum = dft($windowed)
```

applywin

`applywin` — dsp / signal (extended) builtin. Alias for `apply_window`.

```
my $result = applywin $x
# or in a pipeline:
@list |> map applywin |> p
```

approval_voting_max

`approval_voting_max` — batch 43 builtin.

```
my $result = approval_voting_max $x
# or in a pipeline:
@list |> map approval_voting_max |> p
```

approx

`approx_fn` (alias `approx`) — piecewise linear interpolation at query points. Like R's approx().

```
my @y = @{approx([0,1,2], [0,10,0], [0.5, 1.0, 1.5])}
# [5, 10, 5]
```

approx_eq

`approx_eq` — math functions builtin.

```
my $result = approx_eq $x
# or in a pipeline:
@list |> map approx_eq |> p
```

approximate_entropy

`approximate_entropy` — batch 26 builtin.

```
my $result = approximate_entropy $x
# or in a pipeline:
@list |> map approximate_entropy |> p
```

apr_to_apy

`apr_to_apy` — batch 13 builtin.

```
my $result = apr_to_apy $input
```

apsp

`floyd_warshall` (aliases `floydwarshall`, `apsp`) computes all-pairs shortest paths. Takes a distance matrix (use Inf for no edge).

```
my $d = apsp([[0,3,1e18],[1e18,0,1],[1e18,1e18,0]])
```

apy_to_apr

`apy_to_apr` — batch 13 builtin.

```
my $result = apy_to_apr $input
```

ar_model_likelihood

`ar_model_likelihood` — batch 36 builtin.

```
my $result = ar_model_likelihood $x
# or in a pipeline:
@list |> map ar_model_likelihood |> p
```

arange

`arange` — matrix / linear algebra builtin.

```
my $result = arange $x
# or in a pipeline:
@list |> map arange |> p
```

arc_length

`arc_length($radius, $theta)` — computes the arc length of a circular arc. Theta is the central angle in radians. Returns `radius * theta`.

```
p arc_length(10, 3.14159)   # ~31.4 (half circle)
p arc_length(5, 1.57)       # ~7.85 (quarter circle)
```

arch_lm

`arch_lm` — batch 78 builtin.

```
my $result = arch_lm $x
# or in a pipeline:
@list |> map arch_lm |> p
```

arch_lm_test

`arch_lm_test` — batch 36 builtin.

```
my $result = arch_lm_test $x
# or in a pipeline:
@list |> map arch_lm_test |> p
```

arclen

`arclen` — geometry (extended) builtin. Alias for `arc_length`.

```
my $result = arclen $x
# or in a pipeline:
@list |> map arclen |> p
```

arctic_oscillation_step

`arctic_oscillation_step` — batch 42 builtin.

```
my $result = arctic_oscillation_step $x
# or in a pipeline:
@list |> map arctic_oscillation_step |> p
```

area_circle

`area_circle` — geometry / physics builtin.

```
my $result = area_circle $x
# or in a pipeline:
@list |> map area_circle |> p
```

area_ellipse

`area_ellipse` — geometry / physics builtin.

```
my $result = area_ellipse $x
# or in a pipeline:
@list |> map area_ellipse |> p
```

area_rectangle

`area_rectangle` — geometry / physics builtin.

```
my $result = area_rectangle $x
# or in a pipeline:
@list |> map area_rectangle |> p
```

area_trapezoid

`area_trapezoid` — geometry / physics builtin.

```
my $result = area_trapezoid $x
# or in a pipeline:
@list |> map area_trapezoid |> p
```

area_triangle

`area_triangle` — geometry / physics builtin.

```
my $result = area_triangle $x
# or in a pipeline:
@list |> map area_triangle |> p
```

arect

`arect` — geometry / physics builtin. Alias for `area_rectangle`.

```
my $result = arect $x
# or in a pipeline:
@list |> map arect |> p
```

arellano_bond_step

`arellano_bond_step` — batch 36 builtin.

```
my $result = arellano_bond_step $x
# or in a pipeline:
@list |> map arellano_bond_step |> p
```

argc

`argc` — process / env builtin.

```
my $result = argc $x
# or in a pipeline:
@list |> map argc |> p
```

argmax

`argmax` — extended stdlib builtin.

```
my $result = argmax $x
# or in a pipeline:
@list |> map argmax |> p
```

argmin

`argmin` — extended stdlib builtin.

```
my $result = argmin $x
# or in a pipeline:
@list |> map argmin |> p
```

argon2_block_mix

`argon2_block_mix` — batch 26 builtin.

```
my $result = argon2_block_mix $x
# or in a pipeline:
@list |> map argon2_block_mix |> p
```

argpartition

`argpartition` — batch 72 builtin.

```
my $result = argpartition $x
# or in a pipeline:
@list |> map argpartition |> p
```

argsort

`argsort` — extended stdlib builtin.

```
my $result = argsort $x
# or in a pipeline:
@list |> map argsort |> p
```

ari

`ari` — batch 9 builtin. Alias for `automated_readability_index`.

```
my $result = ari $x
# or in a pipeline:
@list |> map ari |> p
```

arias_intensity

`arias_intensity` — batch 65 builtin.

```
my $result = arias_intensity $x
# or in a pipeline:
@list |> map arias_intensity |> p
```

arima_diff

`arima_diff` — batches 17-18 builtin.

```
my $result = arima_diff $x
# or in a pipeline:
@list |> map arima_diff |> p
```

arima_fit

`arima_fit` — batch 78 builtin.

```
my $result = arima_fit $x
# or in a pipeline:
@list |> map arima_fit |> p
```

arima_forecast

`arima_forecast` — batch 78 builtin.

```
my $result = arima_forecast $x
# or in a pipeline:
@list |> map arima_forecast |> p
```

arithmetic_coding_interval

`arithmetic_coding_interval` — batch 40 builtin.

```
my $result = arithmetic_coding_interval $x
# or in a pipeline:
@list |> map arithmetic_coding_interval |> p
```

arithmetic_decode_interval

`arithmetic_decode_interval` — batch 40 builtin.

```
my $result = arithmetic_decode_interval $x
# or in a pipeline:
@list |> map arithmetic_decode_interval |> p
```

arithmetic_sequence

`arithmetic_sequence` — batch 11 builtin.

```
my $result = arithmetic_sequence $x
# or in a pipeline:
@list |> map arithmetic_sequence |> p
```

arithmetic_series

`arithmetic_series($a1, $d, $n)` (alias `arithser`) — sum of n terms of arithmetic sequence starting at a1 with common difference d. Formula: n/2 × (2a1 + (n-1)d).

```
p arithser(1, 1, 100)    # 5050 (1+2+...+100)
p arithser(2, 3, 10)     # 155 (2+5+8+...+29)
```

arity_of

`arity_of` — functional combinators builtin.

```
my $result = arity_of $x
# or in a pipeline:
@list |> map arity_of |> p
```

arma_model_innovation

`arma_model_innovation` — batch 36 builtin.

```
my $result = arma_model_innovation $x
# or in a pipeline:
@list |> map arma_model_innovation |> p
```

arma_order_select

`arma_order_select` — batch 78 builtin.

```
my $result = arma_order_select $x
# or in a pipeline:
@list |> map arma_order_select |> p
```

armstrong_q

`armstrong_q` — batch 11 builtin.

```
my $result = armstrong_q $x
# or in a pipeline:
@list |> map armstrong_q |> p
```

array_agg

`array_agg` — batch 70 builtin.

```
my $result = array_agg $x
# or in a pipeline:
@list |> map array_agg |> p
```

array_difference

`array_difference` — collection (batch 2) builtin.

```
my $result = array_difference $x
# or in a pipeline:
@list |> map array_difference |> p
```

array_intersection

`array_intersection` — collection (batch 2) builtin.

```
my $result = array_intersection $x
# or in a pipeline:
@list |> map array_intersection |> p
```

array_position

`array_position` — batch 70 builtin.

```
my $result = array_position $x
# or in a pipeline:
@list |> map array_position |> p
```

array_positions

`array_positions` — batch 70 builtin.

```
my $result = array_positions $x
# or in a pipeline:
@list |> map array_positions |> p
```

array_remove

`array_remove` — batch 70 builtin.

```
my $result = array_remove $x
# or in a pipeline:
@list |> map array_remove |> p
```

array_replace

`array_replace` — batch 70 builtin.

```
my $result = array_replace $x
# or in a pipeline:
@list |> map array_replace |> p
```

array_to_hstore

array_to_hstore — batch 70 builtin.

```
my $result = array_to_hstore $input
```

array_to_jsonb

array_to_jsonb — batch 70 builtin.

```
my $result = array_to_jsonb $input
```

array_to_string

array_to_string — batch 70 builtin.

```
my $result = array_to_string $input
```

array_union

array_union — collection (batch 2) builtin.

```
my $result = array_union $x
# or in a pipeline:
@list |> map array_union |> p
```

arrhenius_k

arrhenius_k — batch 21 builtin.

```
my $result = arrhenius_k $x
# or in a pipeline:
@list |> map arrhenius_k |> p
```

arrhenius_temp_q10

arrhenius_temp_q10 — batch 12 builtin.

```
my $result = arrhenius_temp_q10 $x
# or in a pipeline:
@list |> map arrhenius_temp_q10 |> p
```

arrow_impossibility_check

arrow_impossibility_check — batch 43 builtin.

```
my $result = arrow_impossibility_check $x
# or in a pipeline:
@list |> map arrow_impossibility_check |> p
```

arrow_independence

arrow_independence — batch 73 builtin.

```
my $result = arrow_independence $x
# or in a pipeline:
@list |> map arrow_independence |> p
```

arrow_pareto_check

`arrow_pareto_check` — batch 43 builtin.

```
my $result = arrow_pareto_check $input
```

articulation_point

`articulation_point` — batch 75 builtin.

```
my $result = articulation_point $x
# or in a pipeline:
@list |> map articulation_point |> p
```

articulation_points

`articulation_points` — batch 13 builtin.

```
my $result = articulation_points $x
# or in a pipeline:
@list |> map articulation_points |> p
```

ascent_parser_step

`ascent_parser_step` — batch 68 builtin.

```
my $result = ascent_parser_step $x
# or in a pipeline:
@list |> map ascent_parser_step |> p
```

ascii

`ascii` — batch 70 builtin.

```
my $result = ascii $x
# or in a pipeline:
@list |> map ascii |> p
```

ascii_chr

`ascii_chr` — string (batch 2) builtin.

```
my $result = ascii_chr $x
# or in a pipeline:
@list |> map ascii_chr |> p
```

ascii_ord

`ascii_ord` — string (batch 2) builtin.

```
my $result = ascii_ord $x
# or in a pipeline:
@list |> map ascii_ord |> p
```

ascii_table

`ascii_table` — batch 9 builtin. Alias for `format_table_simple`.

```
my $result = ascii_table $x
# or in a pipeline:
@list |> map ascii_table |> p
```

asec

`asec` — trig extensions builtin.

```
my $result = asec $x
# or in a pipeline:
@list |> map asec |> p
```

asian_call_mc

`asian_call_mc` — batch 20 builtin.

```
my $result = asian_call_mc $x
# or in a pipeline:
@list |> map asian_call_mc |> p
```

asin

`asin` — trig / math (batch 2) builtin.

```
my $result = asin $x
# or in a pipeline:
@list |> map asin |> p
```

asinh

`asinh` — trig / math (batch 2) builtin.

```
my $result = asinh $x
# or in a pipeline:
@list |> map asinh |> p
```

asrt

`asrt` — extended stdlib builtin. Alias for `argsort`.

```
my $result = asrt $x
# or in a pipeline:
@list |> map asrt |> p
```

assert_type

`assert_type` — conversion / utility (batch 4) builtin.

```
my $result = assert_type $x
# or in a pipeline:
@list |> map assert_type |> p
```

assignment_lower_bound

`assignment_lower_bound` — batch 41 builtin.

```
my $result = assignment_lower_bound $x
# or in a pipeline:
@list |> map assignment_lower_bound |> p
```

assoc

`assoc` — algebraic match builtin.

```
my $result = assoc $x
# or in a pipeline:
@list |> map assoc |> p
```

assoc_fn

`assoc_fn` — go/general functional utilities builtin. Alias for `associate`.

```
my $result = assoc_fn $x
# or in a pipeline:
@list |> map assoc_fn |> p
```

assoc_in

`assoc_in` — algebraic match builtin.

```
my $result = assoc_in $x
# or in a pipeline:
@list |> map assoc_in |> p
```

associate

`associate` — go/general functional utilities builtin.

```
my $result = associate $x
# or in a pipeline:
@list |> map associate |> p
```

astar_search

`astar_search` — batch 75 builtin.

```
my $result = astar_search $x
# or in a pipeline:
@list |> map astar_search |> p
```

astronomical_unit

`astronomical_unit` (alias `au`) — AU ≈ 1.496×10^{11} m. Mean Earth-Sun distance.

```
p au            # 1.495978707e11 m
p au / 1000     # ~149.6 million km
```

at_content

`at_content` — batch 33 builtin.

```
my $result = at_content $x
# or in a pipeline:
@list |> map at_content |> p
```

at_index

`at_index` — javascript array/object methods builtin.

```
my $result = at_index $x
# or in a pipeline:
@list |> map at_index |> p
```

atan

`atan` — trig / math (batch 2) builtin.

```
my $result = atan $x
# or in a pipeline:
@list |> map atan |> p
```

atanh

`atanh` — trig / math (batch 2) builtin.

```
my $result = atanh $x
# or in a pipeline:
@list |> map atanh |> p
```

atbash

`atbash` — string processing (uncategorized batch) builtin.

```
my $result = atbash $x
# or in a pipeline:
@list |> map atbash |> p
```

atbash_cipher

`atbash_cipher` — batches 17-18 builtin.

```
my $result = atbash_cipher $x
# or in a pipeline:
@list |> map atbash_cipher |> p
```

atc

`atc` — color operations builtin. Alias for `ansi_truecolor`.

```
my $result = atc $x
# or in a pipeline:
@list |> map atc |> p
```

ati

`ati` — javascript array/object methods builtin. Alias for `at_index`.

```
my $result = ati $x
# or in a pipeline:
@list |> map ati |> p
```

atime

`atime` — file stat / path builtin. Alias for `file_atime`.

```
my $result = atime $x
# or in a pipeline:
@list |> map atime |> p
```

atm_to_pa

`atm_to_pa` — batch 30 builtin.

```
my $result = atm_to_pa $input
```

atomic_mass_unit

`atomic_mass_unit` — physics constants builtin.

```
my $result = atomic_mass_unit $x
# or in a pipeline:
@list |> map atomic_mass_unit |> p
```

atomic_radius_pm

`atomic_radius_pm` — batch 12 builtin.

```
my $result = atomic_radius_pm $x
# or in a pipeline:
@list |> map atomic_radius_pm |> p
```

atrap

`atrap` — geometry / physics builtin. Alias for `area_trapezoid`.

```
my $result = atrap $x
# or in a pipeline:
@list |> map atrap |> p
```

atri

`atri` — geometry / physics builtin. Alias for `area_triangle`.

```
my $result = atri $x
# or in a pipeline:
@list |> map atri |> p
```

attack_rate

`attack_rate` — batch 61 builtin.

```
my $result = attack_rate $x
# or in a pipeline:
@list |> map attack_rate |> p
```

attempt

`attempt` — functional combinators builtin.

```
my $result = attempt $x
# or in a pipeline:
@list |> map attempt |> p
```

attributable_fraction_pop

`attributable_fraction_pop` — batch 61 builtin.

```
my $result = attributable_fraction_pop $x
# or in a pipeline:
@list |> map attributable_fraction_pop |> p
```

au_to_km

`au_to_km` — batch 55 builtin.

```
my $result = au_to_km $input
```

au_to_m

`au_to_m` — batch 30 builtin.

```
my $result = au_to_m $input
```

augmented_lagrangian_step

`augmented_lagrangian_step` — batch 41 builtin.

```
my $result = augmented_lagrangian_step $x
# or in a pipeline:
@list |> map augmented_lagrangian_step |> p
```

autocorrelation

`autocorrelation` — math / numeric (uncategorized batch) builtin.

```
my $result = autocorrelation $x
# or in a pipeline:
@list |> map autocorrelation |> p
```

automated_readability_index

`automated_readability_index` — batch 9 builtin.

```
my $result = automated_readability_index $x
# or in a pipeline:
@list |> map automated_readability_index |> p
```

average

`average` — additional missing stdlib functions builtin.

```
my $result = average $x
# or in a pipeline:
@list |> map average |> p
```

averageif

`averageif` — batch 57 builtin.

```
my $result = averageif $x
# or in a pipeline:
@list |> map averageif |> p
```

averageifs

`averageifs` — batch 57 builtin.

```
my $result = averageifs $x
# or in a pipeline:
@list |> map averageifs |> p
```

avg

`avg` — pipeline / string helpers builtin.

```
my $result = avg $x
# or in a pipeline:
@list |> map avg |> p
```

avg_degree

`avg_degree` — batch 24 builtin.

```
my $result = avg_degree $x
# or in a pipeline:
@list |> map avg_degree |> p
```

avogadro

`avogadro` — math / physics constants builtin.

```
my $result = avogadro $x
# or in a pipeline:
@list |> map avogadro |> p
```

avogadro_constant

`avogadro_constant` — batch 63 builtin.

```
my $result = avogadro_constant $x
# or in a pipeline:
@list |> map avogadro_constant |> p
```

avogadro_count

`avogadro_count` — batch 21 builtin.

```
my $result = avogadro_count $x
# or in a pipeline:
@list |> map avogadro_count |> p
```

avogadro_number

`avogadro_number` — constants builtin.

```
my $result = avogadro_number $x
# or in a pipeline:
@list |> map avogadro_number |> p
```

avogadros_number

`avogadros_number` — batch 12 builtin.

```
my $result = avogadros_number $x
# or in a pipeline:
@list |> map avogadros_number |> p
```

b2gray

`b2gray` — base / gray code builtin. Alias for `binary_to_gray`.

```
my $result = b2gray $x
# or in a pipeline:
@list |> map b2gray |> p
```

b2i

`b2i` — boolean combinators builtin. Alias for `bool_to_int`.

```
my $result = b2i $x
# or in a pipeline:
@list |> map b2i |> p
```

b64d

b64d — crypto / encoding builtin. Alias for base64_decode.

```
my $result = b64d $x
# or in a pipeline:
@list |> map b64d |> p
```

b64e

b64e — crypto / encoding builtin. Alias for base64_encode.

```
my $result = b64e $x
# or in a pipeline:
@list |> map b64e |> p
```

b_to_kb

b_to_kb — unit conversions builtin. Alias for bytes_to_kb.

```
my $result = b_to_kb $input
```

b_wien

b_wien — physics constants builtin. Alias for wien_constant.

```
my $result = b_wien $x
# or in a pipeline:
@list |> map b_wien |> p
```

babip

babip — batch 56 builtin.

```
my $result = babip $x
# or in a pipeline:
@list |> map babip |> p
```

bac

bac — finance builtin. Alias for bac_estimate.

```
my $result = bac $x
# or in a pipeline:
@list |> map bac |> p
```

bac_estimate

bac_estimate — finance builtin.

```
my $result = bac_estimate $x
# or in a pipeline:
@list |> map bac_estimate |> p
```

bachelier_call

`bachelier_call` — batch 20 builtin.

```
my $result = bachelier_call $x
# or in a pipeline:
@list |> map bachelier_call |> p
```

backward_euler

`backward_euler` — batch 19 builtin.

```
my $result = backward_euler $x
# or in a pipeline:
@list |> map backward_euler |> p
```

badi_year_from_fixed

`badi_year_from_fixed` — batch 52 builtin.

```
my $result = badi_year_from_fixed $x
# or in a pipeline:
@list |> map badi_year_from_fixed |> p
```

bahai_from_fixed

`bahai_from_fixed` — batch 52 builtin.

```
my $result = bahai_from_fixed $x
# or in a pipeline:
@list |> map bahai_from_fixed |> p
```

bairstow

`bairstow` — batch 10 builtin. Alias for `lin_bairstow_step`.

```
my $result = bairstow $x
# or in a pipeline:
@list |> map bairstow |> p
```

balanced_accuracy

`balanced_accuracy` — batch 27 builtin.

```
my $result = balanced_accuracy $x
# or in a pipeline:
@list |> map balanced_accuracy |> p
```

bandpass_filter

`bandpass_filter(\@signal, $low, $high)` (alias `bpf`) — applies a band-pass filter passing frequencies between low and high cutoffs. Combines low-pass and high-pass.

```
my $filtered = bpf(\@signal, 0.1, 0.3)
```

banzhaf_index_two

`banzhaf_index_two` — batch 43 builtin.

```
my $result = banzhaf_index_two $x
# or in a pipeline:
@list |> map banzhaf_index_two |> p
```

bao_scale_today

`bao_scale_today` — batch 31 builtin.

```
my $result = bao_scale_today $x
# or in a pipeline:
@list |> map bao_scale_today |> p
```

bar_to_pascals

`bar_to_pascals` — file stat / path builtin.

```
my $result = bar_to_pascals $input
```

bargaining_set_check

`bargaining_set_check` — batch 43 builtin.

```
my $result = bargaining_set_check $x
# or in a pipeline:
@list |> map bargaining_set_check |> p
```

barrier_up_out_call

`barrier_up_out_call` — batch 20 builtin.

```
my $result = barrier_up_out_call $x
# or in a pipeline:
@list |> map barrier_up_out_call |> p
```

barthann_w

`barthann_w` — batch 74 builtin.

```
my $result = barthann_w $x
# or in a pipeline:
@list |> map barthann_w |> p
```

bartlett_window

`bartlett_window` — batch 25 builtin.

```
my $result = bartlett_window $x
# or in a pipeline:
@list |> map bartlett_window |> p
```

base_convert

`base_convert` — base / gray code builtin.

```
my $result = base_convert $x
# or in a pipeline:
@list |> map base_convert |> p
```

batch

`batch` — collection (batch 2) builtin.

```
my $result = batch $x
# or in a pipeline:
@list |> map batch |> p
```

batched

`batched` — additional missing stdlib functions builtin.

```
my $result = batched $x
# or in a pipeline:
@list |> map batched |> p
```

batchelor_scale_step

`batchelor_scale_step` — batch 42 builtin.

```
my $result = batchelor_scale_step $x
# or in a pipeline:
@list |> map batchelor_scale_step |> p
```

battle_sexes_payoff

`battle_sexes_payoff` — batch 43 builtin.

```
my $result = battle_sexes_payoff $x
# or in a pipeline:
@list |> map battle_sexes_payoff |> p
```

bauer_furuta_step

`bauer_furuta_step` — batch 37 builtin.

```
my $result = bauer_furuta_step $x
# or in a pipeline:
@list |> map bauer_furuta_step |> p
```

bayes_posterior

`bayes_posterior` — batch 33 builtin.

```
my $result = bayes_posterior $x
# or in a pipeline:
@list |> map bayes_posterior |> p
```

bayesian_info_crit

`bayesian_info_crit` — batch 36 builtin.

```
my $result = bayesian_info_crit $x
# or in a pipeline:
@list |> map bayesian_info_crit |> p
```

bayesian_step

`bayesian_step` — batch 78 builtin.

```
my $result = bayesian_step $x
# or in a pipeline:
@list |> map bayesian_step |> p
```

bbox

`bounding_box(@points)` (alias `bbox`) — computes the axis-aligned bounding box of 2D points. Returns [min_x, min_y, max_x, max_y].

```
my @pts = (1,2, 3,4, 0,1, 5,3)
my $bb = bbox(@pts)
p @$bb   # (0, 1, 5, 4)
```

bbox_from_points

`bbox_from_points` — batch 28 builtin.

```
my $result = bbox_from_points $x
# or in a pipeline:
@list |> map bbox_from_points |> p
```

bconv

`bconv` — base / gray code builtin. Alias for `base_convert`.

```
my $result = bconv $x
# or in a pipeline:
@list |> map bconv |> p
```

bcount

`bcount` — extended stdlib batch 3 builtin. Alias for `count_bytes`.

```
my $result = bcount $x
# or in a pipeline:
@list |> map bcount |> p
```

bcrypt_cost_iters

`bcrypt_cost_iters` — batch 26 builtin.

```
my $result = bcrypt_cost_iters $x
# or in a pipeline:
@list |> map bcrypt_cost_iters |> p
```

bdd_apply

`bdd_apply` — batch 67 builtin.

```
my $result = bdd_apply $x
# or in a pipeline:
@list |> map bdd_apply |> p
```

bdd_quantify

`bdd_quantify` — batch 67 builtin.

```
my $result = bdd_quantify $x
# or in a pipeline:
@list |> map bdd_quantify |> p
```

bdd_restrict

`bdd_restrict` — batch 67 builtin.

```
my $result = bdd_restrict $x
# or in a pipeline:
@list |> map bdd_restrict |> p
```

bdf1_step

`bdf1_step` — batch 34 builtin.

```
my $result = bdf1_step $x
# or in a pipeline:
@list |> map bdf1_step |> p
```

bdf2_step

`bdf2_step` — batch 34 builtin.

```
my $result = bdf2_step $x
# or in a pipeline:
@list |> map bdf2_step |> p
```

bdf3_step

`bdf3_step` — batch 34 builtin.

```
my $result = bdf3_step $x
# or in a pipeline:
@list |> map bdf3_step |> p
```

bdf4_step

`bdf4_step` — batch 34 builtin.

```
my $result = bdf4_step $x
# or in a pipeline:
@list |> map bdf4_step |> p
```

bdf5_step

`bdf5_step` — batch 34 builtin.

```
my $result = bdf5_step $x
# or in a pipeline:
@list |> map bdf5_step |> p
```

bdf6_step

`bdf6_step` — batch 34 builtin.

```
my $result = bdf6_step $x
# or in a pipeline:
@list |> map bdf6_step |> p
```

bearing

`bearing` — geometry / physics builtin.

```
my $result = bearing $x
# or in a pipeline:
@list |> map bearing |> p
```

beat_frequency

`beat_frequency` — batch 30 builtin.

```
my $result = beat_frequency $x
# or in a pipeline:
@list |> map beat_frequency |> p
```

beat_to_seconds

`beat_to_seconds` — batch 64 builtin.

```
my $result = beat_to_seconds $input
```

beaufort

`beaufort` — batch 26 builtin.

```
my $result = beaufort $x
# or in a pipeline:
@list |> map beaufort |> p
```

before_n

`before_n` — functional combinators builtin.

```
my $result = before_n $x
# or in a pipeline:
@list |> map before_n |> p
```

bekenstein_entropy_step

`bekenstein_entropy_step` — batch 39 builtin.

```
my $result = bekenstein_entropy_step $x
# or in a pipeline:
@list |> map bekenstein_entropy_step |> p
```

bell_inequality_chsh

`bell_inequality_chsh` — batches 14-16 builtin.

```
my $result = bell_inequality_chsh $x
# or in a pipeline:
@list |> map bell_inequality_chsh |> p
```

bell_number

`bell_number` — math / numeric (uncategorized batch) builtin.

```
my $result = bell_number $x
# or in a pipeline:
@list |> map bell_number |> p
```

bell_state

`bell_state` — batch 80 builtin.

```
my $result = bell_state $x
# or in a pipeline:
@list |> map bell_state |> p
```

bell_state_index

`bell_state_index` — batch 32 builtin.

```
my $result = bell_state_index $x
# or in a pipeline:
@list |> map bell_state_index |> p
```

bellman_equation_step

`bellman_equation_step` — batch 40 builtin.

```
my $result = bellman_equation_step $x
# or in a pipeline:
@list |> map bellman_equation_step |> p
```

bellman_ford

`bellman_ford` (alias `bellmanford`) computes shortest paths from a source in a graph with negative weights. Takes edges [[u,v,w],...], node count, source index.

```
my $d = bellmanford([[0,1,4],[0,2,5],[1,2,-3]], 3, 0)
```

bellman_ford_relax

`bellman_ford_relax` — batch 75 builtin.

```
my $result = bellman_ford_relax $x
# or in a pipeline:
@list |> map bellman_ford_relax |> p
```

belln

`belln` — math / numeric (uncategorized batch) builtin. Alias for `bell_number`.

```
my $result = belln $x
# or in a pipeline:
@list |> map belln |> p
```

benders_decomposition_step

`benders_decomposition_step` — batch 41 builtin.

```
my $result = benders_decomposition_step $x
# or in a pipeline:
@list |> map benders_decomposition_step |> p
```

berger_parker

`berger_parker` — batch 22 builtin.

```
my $result = berger_parker $x
# or in a pipeline:
@list |> map berger_parker |> p
```

bergmann_adjust

`bergmann_adjust` — batch 22 builtin.

```
my $result = bergmann_adjust $x
# or in a pipeline:
@list |> map bergmann_adjust |> p
```

bernoulli

`bernoulli_number` (alias `bernoulli`) computes the nth Bernoulli number. B(0)=1, B(1)=-0.5, odd B(n>1)=0.

```
p bernoulli(0)   # 1
p bernoulli(2)   # 0.1667
p bernoulli(4)   # -0.0333
```

berry_phase_spin_half

`berry_phase_spin_half` — batch 32 builtin.

```
my $result = berry_phase_spin_half $x
# or in a pipeline:
@list |> map berry_phase_spin_half |> p
```

bertrand_eq

`bertrand_eq` — batch 73 builtin.

```
my $result = bertrand_eq $x
# or in a pipeline:
@list |> map bertrand_eq |> p
```

bertrand_price_step

`bertrand_price_step` — batch 43 builtin.

```
my $result = bertrand_price_step $x
# or in a pipeline:
@list |> map bertrand_price_step |> p
```

bessel_in_general

`bessel_in_general` — batch 29 builtin.

```
my $result = bessel_in_general $x
# or in a pipeline:
@list |> map bessel_in_general |> p
```

bessel_jn_general

`bessel_jn_general` — batch 29 builtin.

```
my $result = bessel_jn_general $x
# or in a pipeline:
@list |> map bessel_jn_general |> p
```

bessel_lp

`bessel_lp` — batch 74 builtin.

```
my $result = bessel_lp $x
# or in a pipeline:
@list |> map bessel_lp |> p
```

best_response_dynamic

`best_response_dynamic` — batch 43 builtin.

```
my $result = best_response_dynamic $x
# or in a pipeline:
@list |> map best_response_dynamic |> p
```

beta_pdf

`beta_pdf` (alias `betapdf`) evaluates the Beta distribution PDF at x with shape parameters alpha and beta.

```
p betapdf(0.5, 2, 5)  # Beta(2,5) at x=0.5
```

betti_one

`betti_one` — batch 37 builtin.

```
my $result = betti_one $x
# or in a pipeline:
@list |> map betti_one |> p
```

betti_two

`betti_two` — batch 37 builtin.

```
my $result = betti_two $x
# or in a pipeline:
@list |> map betti_two |> p
```

betti_zero

`betti_zero` — batch 37 builtin.

```
my $result = betti_zero $x
# or in a pipeline:
@list |> map betti_zero |> p
```

between

`between` — conversion / utility (batch 4) builtin.

```
my $result = between $x
# or in a pipeline:
@list |> map between |> p
```

beylkin_wavelet_step

`beylkin_wavelet_step` — batch 40 builtin.

```
my $result = beylkin_wavelet_step $x
# or in a pipeline:
@list |> map beylkin_wavelet_step |> p
```

bfgs_h_update_1d

`bfgs_h_update_1d` — batch 19 builtin.

```
my $result = bfgs_h_update_1d $x
# or in a pipeline:
@list |> map bfgs_h_update_1d |> p
```

bfield_solenoid

`bfield_solenoid` — batch 23 builtin.

```
my $result = bfield_solenoid $x
# or in a pipeline:
@list |> map bfield_solenoid |> p
```

bfield_wire

`bfield_wire` — batch 23 builtin.

```
my $result = bfield_wire $x
# or in a pipeline:
@list |> map bfield_wire |> p
```

bfs

`bfs` — extended stdlib builtin. Alias for `bfs_traverse`.

```
my $result = bfs $x
# or in a pipeline:
@list |> map bfs |> p
```

bfs_count

`bfs_count` — batch 75 builtin.

```
my $result = bfs_count $x
# or in a pipeline:
@list |> map bfs_count |> p
```

bfs_distances

`bfs_distances` — batch 13 builtin.

```
my $result = bfs_distances $x
# or in a pipeline:
@list |> map bfs_distances |> p
```

bfs_traverse

`bfs_traverse` — extended stdlib builtin.

```
my $result = bfs_traverse $x
# or in a pipeline:
@list |> map bfs_traverse |> p
```

bg_black

`bg_black` — color / ansi builtin. Alias for `red`.

```
my $result = bg_black $x
# or in a pipeline:
@list |> map bg_black |> p
```

bg_blue

`bg_blue` — color / ansi builtin. Alias for `red`.

```
my $result = bg_blue $x
# or in a pipeline:
@list |> map bg_blue |> p
```

bg_bright_blue

bg_bright_blue — color / ansi builtin. Alias for red.

```
my $result = bg_bright_blue $x
# or in a pipeline:
@list |> map bg_bright_blue |> p
```

bg_bright_cyan

bg_bright_cyan — color / ansi builtin. Alias for red.

```
my $result = bg_bright_cyan $x
# or in a pipeline:
@list |> map bg_bright_cyan |> p
```

bg_bright_green

bg_bright_green — color / ansi builtin. Alias for red.

```
my $result = bg_bright_green $x
# or in a pipeline:
@list |> map bg_bright_green |> p
```

bg_bright_magenta

bg_bright_magenta — color / ansi builtin. Alias for red.

```
my $result = bg_bright_magenta $x
# or in a pipeline:
@list |> map bg_bright_magenta |> p
```

bg_bright_red

bg_bright_red — color / ansi builtin. Alias for red.

```
my $result = bg_bright_red $x
# or in a pipeline:
@list |> map bg_bright_red |> p
```

bg_bright_white

bg_bright_white — color / ansi builtin. Alias for red.

```
my $result = bg_bright_white $x
# or in a pipeline:
@list |> map bg_bright_white |> p
```

bg_bright_yellow

bg_bright_yellow — color / ansi builtin. Alias for red.

```
my $result = bg_bright_yellow $x
# or in a pipeline:
@list |> map bg_bright_yellow |> p
```

bg_c256

bg_c256 — color / ansi builtin. Alias for bg_color256.

```
my $result = bg_c256 $x
# or in a pipeline:
@list |> map bg_c256 |> p
```

bg_color256

bg_color256 — color / ansi builtin.

```
my $result = bg_color256 $x
# or in a pipeline:
@list |> map bg_color256 |> p
```

bg_cyan

bg_cyan — color / ansi builtin. Alias for red.

```
my $result = bg_cyan $x
# or in a pipeline:
@list |> map bg_cyan |> p
```

bg_green

bg_green — color / ansi builtin. Alias for red.

```
my $result = bg_green $x
# or in a pipeline:
@list |> map bg_green |> p
```

bg_magenta

bg_magenta — color / ansi builtin. Alias for red.

```
my $result = bg_magenta $x
# or in a pipeline:
@list |> map bg_magenta |> p
```

bg_red

bg_red — color / ansi builtin. Alias for red.

```
my $result = bg_red $x
# or in a pipeline:
@list |> map bg_red |> p
```

bg_rgb

bg_rgb — color / ansi builtin.

```
my $result = bg_rgb $x
# or in a pipeline:
@list |> map bg_rgb |> p
```

bg_white

bg_white — color / ansi builtin. Alias for red.

```
my $result = bg_white $x
# or in a pipeline:
@list |> map bg_white |> p
```

bg_yellow

bg_yellow — color / ansi builtin. Alias for red.

```
my $result = bg_yellow $x
# or in a pipeline:
@list |> map bg_yellow |> p
```

bh_adjusted_p

bh_adjusted_p — batch 33 builtin.

```
my $result = bh_adjusted_p $x
# or in a pipeline:
@list |> map bh_adjusted_p |> p
```

bh_entropy

bh_entropy — batch 31 builtin.

```
my $result = bh_entropy $x
# or in a pipeline:
@list |> map bh_entropy |> p
```

bh_evaporation_time

bh_evaporation_time — batch 31 builtin.

```
my $result = bh_evaporation_time $x
# or in a pipeline:
@list |> map bh_evaporation_time |> p
```

bhattacharyya_coefficient

bhattacharyya_coefficient — batch 40 builtin.

```
my $result = bhattacharyya_coefficient $x
# or in a pipeline:
@list |> map bhattacharyya_coefficient |> p
```

bianchi_first_identity_check

bianchi_first_identity_check — batch 39 builtin.

```
my $result = bianchi_first_identity_check $x
# or in a pipeline:
@list |> map bianchi_first_identity_check |> p
```

bianchi_second_identity_check

bianchi_second_identity_check — batch 39 builtin.

```
my $result = bianchi_second_identity_check $x
# or in a pipeline:
@list |> map bianchi_second_identity_check |> p
```

bic

bic — batch 9 builtin.

```
my $result = bic $x
# or in a pipeline:
@list |> map bic |> p
```

biconnected_components

biconnected_components — batch 75 builtin.

```
my $result = biconnected_components $x
# or in a pipeline:
@list |> map biconnected_components |> p
```

bidirectional_dijkstra

`bidirectional_dijkstra` — batch 75 builtin.

```
my $result = bidirectional_dijkstra $x
# or in a pipeline:
@list |> map bidirectional_dijkstra |> p
```

bielliptic_total

`bielliptic_total` — batches 14-16 builtin.

```
my $result = bielliptic_total $x
# or in a pipeline:
@list |> map bielliptic_total |> p
```

bigram

`bigram` — string processing (uncategorized batch) builtin. Alias for `bigrams`.

```
my $result = bigram $x
# or in a pipeline:
@list |> map bigram |> p
```

bigram_perplexity

`bigram_perplexity` — batch 69 builtin.

```
my $result = bigram_perplexity $x
# or in a pipeline:
@list |> map bigram_perplexity |> p
```

bigrams

`bigrams` — string processing (uncategorized batch) builtin.

```
my $result = bigrams $x
# or in a pipeline:
@list |> map bigrams |> p
```

bilinear_xform

`bilinear_xform` — batch 74 builtin.

```
my $result = bilinear_xform $x
# or in a pipeline:
@list |> map bilinear_xform |> p
```

bilinear_zpk_xform

`bilinear_zpk_xform` — batch 74 builtin.

```
my $result = bilinear_zpk_xform $x
# or in a pipeline:
@list |> map bilinear_zpk_xform |> p
```

bin_of

`bin_of` — base conversion builtin. Alias for `to_bin`.

```
my $result = bin_of $x
# or in a pipeline:
@list |> map bin_of |> p
```

bin_packing_best_fit

`bin_packing_best_fit` — batch 41 builtin.

```
my $result = bin_packing_best_fit $x
# or in a pipeline:
@list |> map bin_packing_best_fit |> p
```

bin_packing_first_fit

`bin_packing_first_fit` — batch 41 builtin.

```
my $result = bin_packing_first_fit $x
# or in a pipeline:
@list |> map bin_packing_first_fit |> p
```

bin_packing_lower_bound_l1

`bin_packing_lower_bound_l1` — batch 41 builtin.

```
my $result = bin_packing_lower_bound_l1 $x
# or in a pipeline:
@list |> map bin_packing_lower_bound_l1 |> p
```

bin_packing_next_fit

`bin_packing_next_fit` — batch 41 builtin.

```
my $result = bin_packing_next_fit $x
# or in a pipeline:
@list |> map bin_packing_next_fit |> p
```

bin_to_gray

`bin_to_gray` — batch 10 builtin.

```
my $result = bin_to_gray $input
```

binary_insert

`binary_insert` — matrix operations (uncategorized batch) builtin.

```
my $result = binary_insert $x
# or in a pipeline:
@list |> map binary_insert |> p
```

binary_search

`binary_search` — collection (batch 2) builtin.

```
my $result = binary_search $x
# or in a pipeline:
@list |> map binary_search |> p
```

binary_to_gray

`binary_to_gray` — base / gray code builtin.

```
my $result = binary_to_gray $input
```

bincount

`bincount` — batch 72 builtin.

```
my $result = bincount $x
# or in a pipeline:
@list |> map bincount |> p
```

binder_content_optimal

`binder_content_optimal` — batch 38 builtin.

```
my $result = binder_content_optimal $x
# or in a pipeline:
@list |> map binder_content_optimal |> p
```

binom

`binom` — extended stdlib batch 3 builtin. Alias for `binomial`.

```
my $result = binom $x
# or in a pipeline:
@list |> map binom |> p
```

binom_test

`binom_test` — exact binomial test. Returns two-sided p-value. Like R's binom.test().

```
my $p = binomtest(7, 10, 0.5)  # p-value for 7/10 successes vs p=0.5
```

binomial

`binomial` — extended stdlib batch 3 builtin.

```
my $result = binomial $x
# or in a pipeline:
@list |> map binomial |> p
```

binsert

`binsert` — matrix operations (uncategorized batch) builtin. Alias for `binary_insert`.

```
my $result = binsert $x
# or in a pipeline:
@list |> map binsert |> p
```

biorthogonal_step

`biorthogonal_step` — batch 40 builtin.

```
my $result = biorthogonal_step $x
# or in a pipeline:
@list |> map biorthogonal_step |> p
```

biot_number_step

biot_number_step — batch 42 builtin.

```
my $result = biot_number_step $x
# or in a pipeline:
@list |> map biot_number_step |> p
```

biquad_allpass_coeffs

biquad_allpass_coeffs — batch 25 builtin.

```
my $result = biquad_allpass_coeffs $x
# or in a pipeline:
@list |> map biquad_allpass_coeffs |> p
```

biquad_bandpass_coeffs

biquad_bandpass_coeffs — batch 25 builtin.

```
my $result = biquad_bandpass_coeffs $x
# or in a pipeline:
@list |> map biquad_bandpass_coeffs |> p
```

biquad_highpass_coeffs

biquad_highpass_coeffs — batch 25 builtin.

```
my $result = biquad_highpass_coeffs $x
# or in a pipeline:
@list |> map biquad_highpass_coeffs |> p
```

biquad_highshelf_coeffs

biquad_highshelf_coeffs — batch 25 builtin.

```
my $result = biquad_highshelf_coeffs $x
# or in a pipeline:
@list |> map biquad_highshelf_coeffs |> p
```

biquad_lowpass_coeffs

biquad_lowpass_coeffs — batch 25 builtin.

```
my $result = biquad_lowpass_coeffs $x
# or in a pipeline:
@list |> map biquad_lowpass_coeffs |> p
```

biquad_lowshelf_coeffs

biquad_lowshelf_coeffs — batch 25 builtin.

```
my $result = biquad_lowshelf_coeffs $x
# or in a pipeline:
@list |> map biquad_lowshelf_coeffs |> p
```

biquad_notch_coeffs

biquad_notch_coeffs — batch 25 builtin.

```
my $result = biquad_notch_coeffs $x
# or in a pipeline:
@list |> map biquad_notch_coeffs |> p
```

biquad_peak_coeffs

`biquad_peak_coeffs` — batch 25 builtin.

```
my $result = biquad_peak_coeffs $x
# or in a pipeline:
@list |> map biquad_peak_coeffs |> p
```

biquad_step

`biquad_step` — batch 25 builtin.

```
my $result = biquad_step $x
# or in a pipeline:
@list |> map biquad_step |> p
```

birnbaum_saunders_pdf

`birnbaum_saunders_pdf` — batches 17-18 builtin.

```
my $result = birnbaum_saunders_pdf $x
# or in a pipeline:
@list |> map birnbaum_saunders_pdf |> p
```

bisect

`bisection` (alias `bisect`) finds a root of f(x)=0 in [a,b] via the bisection method. Takes a code ref, a, b, and optional tolerance.

```
my $root = bisect(fn { $_[0]**2 - 2 }, 1, 2)  # √2 ≈ 1.4142
```

bisimulation_step

`bisimulation_step` — batch 67 builtin.

```
my $result = bisimulation_step $x
# or in a pipeline:
@list |> map bisimulation_step |> p
```

bit_and

`bit_and` — bit ops builtin.

```
my $result = bit_and $x
# or in a pipeline:
@list |> map bit_and |> p
```

bit_build

`bit_build` — batch 10 builtin. Alias for `fenwick_build`.

```
my $result = bit_build $x
# or in a pipeline:
@list |> map bit_build |> p
```

bit_clear

`bit_clear` — bit ops builtin.

```
my $result = bit_clear $x
# or in a pipeline:
@list |> map bit_clear |> p
```

bit_flip_prob

`bit_flip_prob` — batch 32 builtin.

```
my $result = bit_flip_prob $x
# or in a pipeline:
@list |> map bit_flip_prob |> p
```

bit_length

`bit_length` — base conversion builtin.

```
my $result = bit_length $x
# or in a pipeline:
@list |> map bit_length |> p
```

bit_not

`bit_not` — bit ops builtin.

```
my $result = bit_not $x
# or in a pipeline:
@list |> map bit_not |> p
```

bit_or

`bit_or` — bit ops builtin.

```
my $result = bit_or $x
# or in a pipeline:
@list |> map bit_or |> p
```

bit_query

`bit_query` — batch 10 builtin. Alias for `fenwick_query`.

```
my $result = bit_query $x
# or in a pipeline:
@list |> map bit_query |> p
```

bit_reverse

`bit_reverse` — batch 10 builtin. Alias for `bit_reverse_32`.

```
my $result = bit_reverse $x
# or in a pipeline:
@list |> map bit_reverse |> p
```

bit_reverse_32

`bit_reverse_32` — batch 10 builtin.

```
my $result = bit_reverse_32 $x
# or in a pipeline:
@list |> map bit_reverse_32 |> p
```

bit_reverse_64

`bit_reverse_64` — batch 35 builtin.

```
my $result = bit_reverse_64 $x
# or in a pipeline:
@list |> map bit_reverse_64 |> p
```

bit_set

`bit_set` — bit ops builtin.

```
my $result = bit_set $x
# or in a pipeline:
@list |> map bit_set |> p
```

bit_test

`bit_test` — bit ops builtin.

```
my $result = bit_test $x
# or in a pipeline:
@list |> map bit_test |> p
```

bit_toggle

`bit_toggle` — bit ops builtin.

```
my $result = bit_toggle $x
# or in a pipeline:
@list |> map bit_toggle |> p
```

bit_xor

`bit_xor` — bit ops builtin.

```
my $result = bit_xor $x
# or in a pipeline:
@list |> map bit_xor |> p
```

bitcount

`bitcount` — batch 71 builtin.

```
my $result = bitcount $x
# or in a pipeline:
@list |> map bitcount |> p
```

bitlen

`bitlen` — base conversion builtin. Alias for `bit_length`.

```
my $result = bitlen $x
# or in a pipeline:
@list |> map bitlen |> p
```

bitop

`bitop` — batch 71 builtin.

```
my $result = bitop $x
# or in a pipeline:
@list |> map bitop |> p
```

bitpos

`bitpos` — batch 71 builtin.

```
my $result = bitpos $x
# or in a pipeline:
@list |> map bitpos |> p
```

bits_count

`bits_count` — base conversion builtin.

```
my $result = bits_count $x
# or in a pipeline:
@list |> map bits_count |> p
```

bits_to_bytes

`bits_to_bytes` — unit conversions builtin.

```
my $result = bits_to_bytes $input
```

bitset_clear

`bitset_clear` — data structure helpers builtin.

```
my $result = bitset_clear $x
# or in a pipeline:
@list |> map bitset_clear |> p
```

bitset_new

`bitset_new` — data structure helpers builtin.

```
my $result = bitset_new $x
# or in a pipeline:
@list |> map bitset_new |> p
```

bitset_set

`bitset_set` — data structure helpers builtin.

```
my $result = bitset_set $x
# or in a pipeline:
@list |> map bitset_set |> p
```

bitset_test

`bitset_test` — data structure helpers builtin.

```
my $result = bitset_test $x
# or in a pipeline:
@list |> map bitset_test |> p
```

bkt

`bkt` — matrix operations (uncategorized batch) builtin. Alias for `bucket`.

```
my $result = bkt $x
# or in a pipeline:
@list |> map bkt |> p
```

bl

`bl` — algebraic match builtin. Alias for `butlast`.

```
my $result = bl $x
# or in a pipeline:
@list |> map bl |> p
```

black

`black` — color / ansi builtin. Alias for `red`.

```
my $result = black $x
# or in a pipeline:
@list |> map black |> p
```

black76_call

`black76_call` — batch 20 builtin.

```
my $result = black76_call $x
# or in a pipeline:
@list |> map black76_call |> p
```

black_karasinski_drift

`black_karasinski_drift` — batch 20 builtin.

```
my $result = black_karasinski_drift $x
# or in a pipeline:
@list |> map black_karasinski_drift |> p
```

black_method_winner

`black_method_winner` — batch 43 builtin.

```
my $result = black_method_winner $x
# or in a pipeline:
@list |> map black_method_winner |> p
```

black_scholes_call

`black_scholes_call($S, $K, $r, $T, $sigma)` (alias `bscall`) — computes Black-Scholes price for a European call option. S=spot, K=strike, r=rate, T=time to expiry, sigma=volatility.

```
p bscall(100, 100, 0.05, 1, 0.2)   # ~10.45
```

black_scholes_put

`black_scholes_put($S, $K, $r, $T, $sigma)` (alias `bsput`) — computes Black-Scholes price for a European put option.

```
p bsput(100, 100, 0.05, 1, 0.2)   # ~5.57
```

blackman

`window_blackman($n)` (alias `blackman`) — generates a Blackman window of length n. Provides excellent sidelobe suppression at the cost of main lobe width.

```
my $w = blackman(1024)
```

blackman_harris_window

`blackman_harris_window` — batch 25 builtin.

```
my $result = blackman_harris_window $x
# or in a pipeline:
@list |> map blackman_harris_window |> p
```

blackman_w

`blackman_w` — batch 74 builtin.

```
my $result = blackman_w $x
# or in a pipeline:
@list |> map blackman_w |> p
```

blackman_window

`blackman_window` — batch 25 builtin.

```
my $result = blackman_window $x
# or in a pipeline:
@list |> map blackman_window |> p
```

blackscholes_call

`blackscholes_call` — batch 20 builtin. Alias for `bs_call`.

```
my $result = blackscholes_call $x
# or in a pipeline:
@list |> map blackscholes_call |> p
```

blackscholes_put

`blackscholes_put` — batch 20 builtin. Alias for `bs_put`.

```
my $result = blackscholes_put $x
# or in a pipeline:
@list |> map blackscholes_put |> p
```

blessed

`blessed` — list / aggregate builtin.

```
my $result = blessed $x
# or in a pipeline:
@list |> map blessed |> p
```

blink

`blink` — color / ansi builtin. Alias for `red`.

```
my $result = blink $x
# or in a pipeline:
@list |> map blink |> p
```

bloch_purity_check

`bloch_purity_check` — batch 32 builtin.

```
my $result = bloch_purity_check $x
# or in a pipeline:
@list |> map bloch_purity_check |> p
```

bloch_sphere_x

`bloch_sphere_x` — batch 80 builtin.

```
my $result = bloch_sphere_x $x
# or in a pipeline:
@list |> map bloch_sphere_x |> p
```

bloch_sphere_z

`bloch_sphere_z` — batch 80 builtin.

```
my $result = bloch_sphere_z $x
# or in a pipeline:
@list |> map bloch_sphere_z |> p
```

bloch_to_density_real

`bloch_to_density_real` — batch 32 builtin.

```
my $result = bloch_to_density_real $input
```

block_devices

`block_devices` — filesystem extensions builtin.

```
my $result = block_devices $x
# or in a pipeline:
@list |> map block_devices |> p
```

blossom_match_step

`blossom_match_step` — batch 41 builtin.

```
my $result = blossom_match_step $x
# or in a pipeline:
@list |> map blossom_match_step |> p
```

blosum45_score

`blosum45_score` — batch 33 builtin.

```
my $result = blosum45_score $x
# or in a pipeline:
@list |> map blosum45_score |> p
```

blue

`blue` — color / ansi builtin. Alias for `red`.

```
my $result = blue $x
# or in a pipeline:
@list |> map blue |> p
```

blue_bold

`blue_bold` — color / ansi builtin. Alias for `red`.

```
my $result = blue_bold $x
# or in a pipeline:
@list |> map blue_bold |> p
```

bm_horspool

bm_horspool — batch 10 builtin. Alias for horspool_search.

```
my $result = bm_horspool $x
# or in a pipeline:
@list |> map bm_horspool |> p
```

bmi

bmi — finance builtin.

```
my $result = bmi $x
# or in a pipeline:
@list |> map bmi |> p
```

bmi_calc

bmi_calc — physics formulas builtin.

```
my $result = bmi_calc $x
# or in a pipeline:
@list |> map bmi_calc |> p
```

bmr_harris_benedict_female

bmr_harris_benedict_female — batch 12 builtin.

```
my $result = bmr_harris_benedict_female $x
# or in a pipeline:
@list |> map bmr_harris_benedict_female |> p
```

bmr_harris_benedict_male

bmr_harris_benedict_male — batch 12 builtin.

```
my $result = bmr_harris_benedict_male $x
# or in a pipeline:
@list |> map bmr_harris_benedict_male |> p
```

bode_gain_margin

bode_gain_margin — batch 59 builtin.

```
my $result = bode_gain_margin $x
# or in a pipeline:
@list |> map bode_gain_margin |> p
```

bode_phase

bode_phase — batch 6 builtin. Alias for bode_phase_deg.

```
my $result = bode_phase $x
# or in a pipeline:
@list |> map bode_phase |> p
```

bode_phase_margin

bode_phase_margin — batch 59 builtin.

```
my $result = bode_phase_margin $x
# or in a pipeline:
@list |> map bode_phase_margin |> p
```

body_surface_area_dubois

body_surface_area_dubois — batch 12 builtin.

```
my $result = body_surface_area_dubois $x
# or in a pipeline:
@list |> map body_surface_area_dubois |> p
```

body_wave_mb

body_wave_mb — batch 65 builtin.

```
my $result = body_wave_mb $x
# or in a pipeline:
@list |> map body_wave_mb |> p
```

bogacki_shampine_step

bogacki_shampine_step — batch 34 builtin.

```
my $result = bogacki_shampine_step $x
# or in a pipeline:
@list |> map bogacki_shampine_step |> p
```

bohr_energy_ev

bohr_energy_ev — batch 21 builtin.

```
my $result = bohr_energy_ev $x
# or in a pipeline:
@list |> map bohr_energy_ev |> p
```

bohr_magneton

bohr_magneton — physics constants builtin.

```
my $result = bohr_magneton $x
# or in a pipeline:
@list |> map bohr_magneton |> p
```

bohr_radius_n

bohr_radius_n — batch 21 builtin.

```
my $result = bohr_radius_n $x
# or in a pipeline:
@list |> map bohr_radius_n |> p
```

bold

bold — color / ansi builtin. Alias for red.

```
my $result = bold $x
# or in a pipeline:
@list |> map bold |> p
```

bold_blue

bold_blue — color / ansi builtin. Alias for red.

```
my $result = bold_blue $x
# or in a pipeline:
@list |> map bold_blue |> p
```

bold_cyan

bold_cyan — color / ansi builtin. Alias for red.

```
my $result = bold_cyan $x
# or in a pipeline:
@list |> map bold_cyan |> p
```

bold_green

bold_green — color / ansi builtin. Alias for red.

```
my $result = bold_green $x
# or in a pipeline:
@list |> map bold_green |> p
```

bold_magenta

bold_magenta — color / ansi builtin. Alias for red.

```
my $result = bold_magenta $x
# or in a pipeline:
@list |> map bold_magenta |> p
```

bold_red

bold_red — color / ansi builtin. Alias for red.

```
my $result = bold_red $x
# or in a pipeline:
@list |> map bold_red |> p
```

bold_white

bold_white — color / ansi builtin. Alias for red.

```
my $result = bold_white $x
# or in a pipeline:
@list |> map bold_white |> p
```

bold_yellow

bold_yellow — color / ansi builtin. Alias for red.

```
my $result = bold_yellow $x
# or in a pipeline:
@list |> map bold_yellow |> p
```

boltz

boltz — constants builtin. Alias for boltzmann_constant.

```
my $result = boltz $x
# or in a pipeline:
@list |> map boltz |> p
```

boltzmann

boltzmann — math / physics constants builtin.

```
my $result = boltzmann $x
# or in a pipeline:
@list |> map boltzmann |> p
```

boltzmann_const

`boltzmann_const` — batch 12 builtin.

```
my $result = boltzmann_const $x
# or in a pipeline:
@list |> map boltzmann_const |> p
```

boltzmann_constant

`boltzmann_constant` — constants builtin.

```
my $result = boltzmann_constant $x
# or in a pipeline:
@list |> map boltzmann_constant |> p
```

boltzmann_softmax_action

`boltzmann_softmax_action` — batch 40 builtin.

```
my $result = boltzmann_softmax_action $x
# or in a pipeline:
@list |> map boltzmann_softmax_action |> p
```

bond_duration

`duration($face, $coupon_rate, $ytm, $periods)` (alias `bond_duration`) — computes the Macaulay duration of a bond, measuring interest rate sensitivity as weighted average time to receive cash flows.

```
p duration(1000, 0.05, 0.05, 10)   # ~7.7 years
```

bond_price

`bond_price($face, $coupon_rate, $ytm, $periods)` — computes the present value (price) of a bond given face value, coupon rate, yield to maturity, and number of periods.

```
p bond_price(1000, 0.05, 0.04, 10)   # ~1081 (premium)
p bond_price(1000, 0.05, 0.06, 10)   # ~926 (discount)
```

bond_yield

`bond_yield($price, $face, $coupon_rate, $periods)` — computes the yield to maturity of a bond given its current price. Uses Newton-Raphson iteration.

```
p bond_yield(950, 1000, 0.05, 10)   # ~5.7%
```

bondi_mass_step

`bondi_mass_step` — batch 39 builtin.

```
my $result = bondi_mass_step $x
# or in a pipeline:
@list |> map bondi_mass_step |> p
```

bondprc

`bondprc` — finance (extended) builtin. Alias for `bond_price`.

```
my $result = bondprc $x
# or in a pipeline:
@list |> map bondprc |> p
```

bondyld

`bondyld` — finance (extended) builtin. Alias for `bond_yield`.

```
my $result = bondyld $x
# or in a pipeline:
@list |> map bondyld |> p
```

bool_each

`bool_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = bool_each $x
# or in a pipeline:
@list |> map bool_each |> p
```

bool_to_int

`bool_to_int` — boolean combinators builtin.

```
my $result = bool_to_int $input
```

boole_int

`boole_int` — batch 19 builtin. Alias for `boole_rule`.

```
my $result = boole_int $x
# or in a pipeline:
@list |> map boole_int |> p
```

boole_rule

`boole_rule` — batch 19 builtin.

```
my $result = boole_rule $x
# or in a pipeline:
@list |> map boole_rule |> p
```

bootstrap_se_estimate

`bootstrap_se_estimate` — batch 36 builtin.

```
my $result = bootstrap_se_estimate $x
# or in a pipeline:
@list |> map bootstrap_se_estimate |> p
```

borda_count_step

`borda_count_step` — batch 43 builtin.

```
my $result = borda_count_step $x
# or in a pipeline:
@list |> map borda_count_step |> p
```

borders_of_string

borders_of_string — batches 14-16 builtin.

```
my $result = borders_of_string $x
# or in a pipeline:
@list |> map borders_of_string |> p
```

bornhuetter_ferguson

bornhuetter_ferguson — batch 60 builtin.

```
my $result = bornhuetter_ferguson $x
# or in a pipeline:
@list |> map bornhuetter_ferguson |> p
```

boruvka_step

boruvka_step — batch 75 builtin.

```
my $result = boruvka_step $x
# or in a pipeline:
@list |> map boruvka_step |> p
```

bose_einstein_g

bose_einstein_g — batch 29 builtin.

```
my $result = bose_einstein_g $x
# or in a pipeline:
@list |> map bose_einstein_g |> p
```

bose_einstein_pmf

bose_einstein_pmf — batch 32 builtin.

```
my $result = bose_einstein_pmf $x
# or in a pipeline:
@list |> map bose_einstein_pmf |> p
```

boston_mechanism_step

boston_mechanism_step — batch 43 builtin.

```
my $result = boston_mechanism_step $x
# or in a pipeline:
@list |> map boston_mechanism_step |> p
```

both

both — boolean combinators builtin.

```
my $result = both $x
# or in a pipeline:
@list |> map both |> p
```

bowley_skewness

bowley_skewness — batches 14-16 builtin.

```
my $result = bowley_skewness $x
# or in a pipeline:
@list |> map bowley_skewness |> p
```

box_blur_kernel

box_blur_kernel — batches 14-16 builtin.

```
my $result = box_blur_kernel $x
# or in a pipeline:
@list |> map box_blur_kernel |> p
```

box_pierce_test

box_pierce_test — batch 36 builtin.

```
my $result = box_pierce_test $x
# or in a pipeline:
@list |> map box_pierce_test |> p
```

boyer_lindquist_step

boyer_lindquist_step — batch 39 builtin.

```
my $result = boyer_lindquist_step $x
# or in a pipeline:
@list |> map boyer_lindquist_step |> p
```

boykov_kolmogorov_step

boykov_kolmogorov_step — batch 41 builtin.

```
my $result = boykov_kolmogorov_step $x
# or in a pipeline:
@list |> map boykov_kolmogorov_step |> p
```

bp_elevation

bp_elevation — batch 21 builtin.

```
my $result = bp_elevation $x
# or in a pipeline:
@list |> map bp_elevation |> p
```

bpm_to_midi_tick_us

bpm_to_midi_tick_us — batch 30 builtin.

```
my $result = bpm_to_midi_tick_us $input
```

bpm_to_period

bpm_to_period — batch 9 builtin.

```
my $result = bpm_to_period $input
```

bpm_to_spb

bpm_to_spb — batch 30 builtin.

```
my $result = bpm_to_spb $input
```

bracelet_count

`bracelet_count` — batch 9 builtin.

```
my $result = bracelet_count $x
# or in a pipeline:
@list |> map bracelet_count |> p
```

bracket

`bracket` — string helpers (batch 4) builtin.

```
my $result = bracket $x
# or in a pipeline:
@list |> map bracket |> p
```

braille

`braille` — encoding / phonetics builtin. Alias for `braille_encode`.

```
my $result = braille $x
# or in a pipeline:
@list |> map braille |> p
```

braille_encode

`braille_encode` — encoding / phonetics builtin.

```
my $result = braille_encode $x
# or in a pipeline:
@list |> map braille_encode |> p
```

branch_and_bound_step

`branch_and_bound_step` — batch 41 builtin.

```
my $result = branch_and_bound_step $x
# or in a pipeline:
@list |> map branch_and_bound_step |> p
```

branch_length_substitutions

`branch_length_substitutions` — batch 33 builtin.

```
my $result = branch_length_substitutions $x
# or in a pipeline:
@list |> map branch_length_substitutions |> p
```

brans_dicke_step

`brans_dicke_step` — batch 39 builtin.

```
my $result = brans_dicke_step $x
# or in a pipeline:
@list |> map brans_dicke_step |> p
```

brayton_efficiency

`brayton_efficiency` — batch 12 builtin.

```
my $result = brayton_efficiency $x
# or in a pipeline:
@list |> map brayton_efficiency |> p
```

break_even

`break_even` — finance builtin.

```
my $result = break_even $x
# or in a pipeline:
@list |> map break_even |> p
```

break_fn

`break_fn` — haskell list functions builtin.

```
my $result = break_fn $x
# or in a pipeline:
@list |> map break_fn |> p
```

breakeven

`breakeven` — finance builtin. Alias for `break_even`.

```
my $result = breakeven $x
# or in a pipeline:
@list |> map breakeven |> p
```

brent

`brent` — batch 19 builtin. Alias for `brent_root`.

```
my $result = brent $x
# or in a pipeline:
@list |> map brent |> p
```

brent_root

`brent_root` — batch 19 builtin.

```
my $result = brent_root $x
# or in a pipeline:
@list |> map brent_root |> p
```

breusch_godfrey

`breusch_godfrey` — batch 78 builtin.

```
my $result = breusch_godfrey $x
# or in a pipeline:
@list |> map breusch_godfrey |> p
```

breusch_godfrey_test

`breusch_godfrey_test` — batch 36 builtin.

```
my $result = breusch_godfrey_test $x
# or in a pipeline:
@list |> map breusch_godfrey_test |> p
```

breusch_pagan_lm

`breusch_pagan_lm` — batch 36 builtin.

```
my $result = breusch_pagan_lm $x
# or in a pipeline:
@list |> map breusch_pagan_lm |> p
```

breusch_pagan_test

`breusch_pagan_test` — batch 36 builtin.

```
my $result = breusch_pagan_test $x
# or in a pipeline:
@list |> map breusch_pagan_test |> p
```

brewster

`brewster` — physics formulas builtin. Alias for `brewster_angle`.

```
my $result = brewster $x
# or in a pipeline:
@list |> map brewster |> p
```

brewster_angle

`brewster_angle` — physics formulas builtin.

```
my $result = brewster_angle $x
# or in a pipeline:
@list |> map brewster_angle |> p
```

bridge_edge

`bridge_edge` — batch 75 builtin.

```
my $result = bridge_edge $x
# or in a pipeline:
@list |> map bridge_edge |> p
```

bridges

`bridges` — batch 24 builtin.

```
my $result = bridges $x
# or in a pipeline:
@list |> map bridges |> p
```

bridges_edges

`bridges_edges` — batch 13 builtin.

```
my $result = bridges_edges $x
# or in a pipeline:
@list |> map bridges_edges |> p
```

brier_score

`brier_score` — batch 27 builtin.

```
my $result = brier_score $x
# or in a pipeline:
@list |> map brier_score |> p
```

bright_blue

`bright_blue` — color / ansi builtin. Alias for `red`.

```
my $result = bright_blue $x
# or in a pipeline:
@list |> map bright_blue |> p
```

bright_cyan

`bright_cyan` — color / ansi builtin. Alias for `red`.

```
my $result = bright_cyan $x
# or in a pipeline:
@list |> map bright_cyan |> p
```

bright_green

`bright_green` — color / ansi builtin. Alias for `red`.

```
my $result = bright_green $x
# or in a pipeline:
@list |> map bright_green |> p
```

bright_magenta

`bright_magenta` — color / ansi builtin. Alias for `red`.

```
my $result = bright_magenta $x
# or in a pipeline:
@list |> map bright_magenta |> p
```

bright_red

`bright_red` — color / ansi builtin. Alias for `red`.

```
my $result = bright_red $x
# or in a pipeline:
@list |> map bright_red |> p
```

bright_white

`bright_white` — color / ansi builtin. Alias for `red`.

```
my $result = bright_white $x
# or in a pipeline:
@list |> map bright_white |> p
```

bright_yellow

`bright_yellow` — color / ansi builtin. Alias for `red`.

```
my $result = bright_yellow $x
# or in a pipeline:
@list |> map bright_yellow |> p
```

brkf

`brkf` — haskell list functions builtin. Alias for `break_fn`.

```
my $result = brkf $x
# or in a pipeline:
@list |> map brkf |> p
```

bron_kerbosch

`bron_kerbosch` — batch 75 builtin.

```
my $result = bron_kerbosch $x
# or in a pipeline:
@list |> map bron_kerbosch |> p
```

brotli_distance_code_count

`brotli_distance_code_count` — batch 40 builtin.

```
my $result = brotli_distance_code_count $x
# or in a pipeline:
@list |> map brotli_distance_code_count |> p
```

brotli_encode_meta

`brotli_encode_meta` — batch 81 builtin.

```
my $result = brotli_encode_meta $x
# or in a pipeline:
@list |> map brotli_encode_meta |> p
```

brotli_huffman_table

`brotli_huffman_table` — batch 62 builtin.

```
my $result = brotli_huffman_table $x
# or in a pipeline:
@list |> map brotli_huffman_table |> p
```

brotli_meta_block

`brotli_meta_block` — batch 62 builtin.

```
my $result = brotli_meta_block $x
# or in a pipeline:
@list |> map brotli_meta_block |> p
```

brown_york_quasilocal

`brown_york_quasilocal` — batch 39 builtin.

```
my $result = brown_york_quasilocal $x
# or in a pipeline:
@list |> map brown_york_quasilocal |> p
```

brunt_vaisala_full

`brunt_vaisala_full` — batch 42 builtin.

```
my $result = brunt_vaisala_full $x
# or in a pipeline:
@list |> map brunt_vaisala_full |> p
```

brzozowski_derivative

brzozowski_derivative — batch 68 builtin.

```
my $result = brzozowski_derivative $x
# or in a pipeline:
@list |> map brzozowski_derivative |> p
```

bs_call

bs_call — batch 20 builtin.

```
my $result = bs_call $x
# or in a pipeline:
@list |> map bs_call |> p
```

bs_delta

bs_delta (aliases bsdelta, option_delta) computes the Black-Scholes delta (∂C/∂S). Args: S, K, T, r, sigma.

```
p bsdelta(100, 100, 1, 0.05, 0.2)  # ≈ 0.64
```

bs_gamma

bs_gamma computes the Black-Scholes gamma (∂²C/∂S²). Measures convexity of option value.

```
p bsgamma(100, 100, 1, 0.05, 0.2)  # ≈ 0.019
```

bs_put

bs_put — batch 20 builtin.

```
my $result = bs_put $x
# or in a pipeline:
@list |> map bs_put |> p
```

bs_rho

bs_rho computes the Black-Scholes rho (∂C/∂r). Sensitivity to interest rate.

```
p bsrho(100, 100, 1, 0.05, 0.2)  # ≈ 46
```

bs_rho_call

bs_rho_call — batch 20 builtin.

```
my $result = bs_rho_call $x
# or in a pipeline:
@list |> map bs_rho_call |> p
```

bs_theta

bs_theta computes the Black-Scholes theta (∂C/∂t). Time decay per unit time.

```
p bstheta(100, 100, 1, 0.05, 0.2)  # negative (time decay)
```

bs_theta_call

bs_theta_call — batch 20 builtin.

```
my $result = bs_theta_call $x
# or in a pipeline:
@list |> map bs_theta_call |> p
```

bs_vega

`bs_vega` computes the Black-Scholes vega (∂C/∂σ). Sensitivity to volatility.

```
p bsvega(100, 100, 1, 0.05, 0.2)  # ≈ 37.5
```

bsa_dubois

`bsa_dubois` — batch 12 builtin. Alias for `body_surface_area_dubois`.

```
my $result = bsa_dubois $x
# or in a pipeline:
@list |> map bsa_dubois |> p
```

bsearch

`bsearch` — collection (batch 2) builtin. Alias for `binary_search`.

```
my $result = bsearch $x
# or in a pipeline:
@list |> map bsearch |> p
```

bsgs_discrete_log

`bsgs_discrete_log` — batch 35 builtin.

```
my $result = bsgs_discrete_log $x
# or in a pipeline:
@list |> map bsgs_discrete_log |> p
```

bsize

`bsize` — misc / utility builtin. Alias for `byte_size`.

```
my $result = bsize $x
# or in a pipeline:
@list |> map bsize |> p
```

btc_addr_check

`btc_addr_check` — batch 82 builtin.

```
my $result = btc_addr_check $x
# or in a pipeline:
@list |> map btc_addr_check |> p
```

btch

`btch` — additional missing stdlib functions builtin. Alias for `batched`.

```
my $result = btch $x
# or in a pipeline:
@list |> map btch |> p
```

btrim

`btrim` — batch 70 builtin.

```
my $result = btrim $x
# or in a pipeline:
@list |> map btrim |> p
```

btu_to_joules

`btu_to_joules` — batch 30 builtin.

```
my $result = btu_to_joules $input
```

bucket

`bucket` — matrix operations (uncategorized batch) builtin.

```
my $result = bucket $x
# or in a pipeline:
@list |> map bucket |> p
```

buffer_capacity_acid_base

`buffer_capacity_acid_base` — batch 38 builtin.

```
my $result = buffer_capacity_acid_base $x
# or in a pipeline:
@list |> map buffer_capacity_acid_base |> p
```

bulirsch_stoer_step

`bulirsch_stoer_step` — batch 34 builtin.

```
my $result = bulirsch_stoer_step $x
# or in a pipeline:
@list |> map bulirsch_stoer_step |> p
```

burr_xii_cdf

`burr_xii_cdf` — batches 17-18 builtin.

```
my $result = burr_xii_cdf $x
# or in a pipeline:
@list |> map burr_xii_cdf |> p
```

burr_xii_pdf

`burr_xii_pdf` — batches 17-18 builtin.

```
my $result = burr_xii_pdf $x
# or in a pipeline:
@list |> map burr_xii_pdf |> p
```

business_days_between

`business_days_between` — batch 13 builtin.

```
my $result = business_days_between $x
# or in a pipeline:
@list |> map business_days_between |> p
```

butlast

`butlast` — algebraic match builtin.

```
my $result = butlast $x
# or in a pipeline:
@list |> map butlast |> p
```

butler_volmer_current

`butler_volmer_current` — batch 38 builtin.

```
my $result = butler_volmer_current $x
# or in a pipeline:
@list |> map butler_volmer_current |> p
```

butter_hp_mag

`butter_hp_mag` — batch 74 builtin.

```
my $result = butter_hp_mag $x
# or in a pipeline:
@list |> map butter_hp_mag |> p
```

butter_lp_re

`butter_lp_re` — batch 74 builtin.

```
my $result = butter_lp_re $x
# or in a pipeline:
@list |> map butter_lp_re |> p
```

butterworth_order

`butterworth_order` — batch 25 builtin.

```
my $result = butterworth_order $x
# or in a pipeline:
@list |> map butterworth_order |> p
```

butterworth_prewarp

`butterworth_prewarp` — batch 25 builtin.

```
my $result = butterworth_prewarp $x
# or in a pipeline:
@list |> map butterworth_prewarp |> p
```

bwt_decode

`bwt_decode` — batches 14-16 builtin.

```
my $result = bwt_decode $x
# or in a pipeline:
@list |> map bwt_decode |> p
```

bwt_encode

`bwt_encode` — batches 14-16 builtin.

```
my $result = bwt_encode $x
# or in a pipeline:
@list |> map bwt_encode |> p
```

byte_length

byte_length — string helpers (batch 4) builtin.

```
my $result = byte_length $x
# or in a pipeline:
@list |> map byte_length |> p
```

byte_size

byte_size — misc / utility builtin.

```
my $result = byte_size $x
# or in a pipeline:
@list |> map byte_size |> p
```

bytecode_disasm_step

bytecode_disasm_step — batch 68 builtin.

```
my $result = bytecode_disasm_step $x
# or in a pipeline:
@list |> map bytecode_disasm_step |> p
```

bytes_to_bits

bytes_to_bits — unit conversions builtin.

```
my $result = bytes_to_bits $input
```

bytes_to_gb

bytes_to_gb — unit conversions builtin.

```
my $result = bytes_to_gb $input
```

bytes_to_hex_str

bytes_to_hex_str — string helpers (batch 4) builtin.

```
my $result = bytes_to_hex_str $input
```

bytes_to_kb

bytes_to_kb — unit conversions builtin.

```
my $result = bytes_to_kb $input
```

bytes_to_mb

bytes_to_mb — unit conversions builtin.

```
my $result = bytes_to_mb $input
```

bz2_encode_step

`bz2_encode_step` — batch 81 builtin.

```
my $result = bz2_encode_step $x
# or in a pipeline:
@list |> map bz2_encode_step |> p
```

c256

`c256` — color / ansi builtin. Alias for `color256`.

```
my $result = c256 $x
# or in a pipeline:
@list |> map c256 |> p
```

c2s

`c2s` — string processing (uncategorized batch) builtin. Alias for `camel_to_snake`.

```
my $result = c2s $x
# or in a pipeline:
@list |> map c2s |> p
```

c_parallel

`c_parallel` — batch 12 builtin. Alias for `capacitance_parallel`.

```
my $result = c_parallel $x
# or in a pipeline:
@list |> map c_parallel |> p
```

c_series

`c_series` — batch 12 builtin. Alias for `capacitance_series`.

```
my $result = c_series $x
# or in a pipeline:
@list |> map c_series |> p
```

c_to_f

`c_to_f` — unit conversions builtin.

```
my $result = c_to_f $input
```

c_to_k

`c_to_k` — unit conversions builtin.

```
my $result = c_to_k $input
```

cabling_pair_signature

`cabling_pair_signature` — batch 37 builtin.

```
my $result = cabling_pair_signature $x
# or in a pipeline:
@list |> map cabling_pair_signature |> p
```

cache_clear

`cache_clear` — caching builtin.

```
my $result = cache_clear $x
# or in a pipeline:
@list |> map cache_clear |> p
```

cache_control_parse

`cache_control_parse` — batch 81 builtin.

```
my $result = cache_control_parse $x
# or in a pipeline:
@list |> map cache_control_parse |> p
```

cache_exists

`cache_exists` — caching builtin.

```
my $result = cache_exists $x
# or in a pipeline:
@list |> map cache_exists |> p
```

cache_stats

`cache_stats` — caching builtin.

```
my $result = cache_stats $x
# or in a pipeline:
@list |> map cache_stats |> p
```

cacheview

`cacheview` — caching builtin.

```
my $result = cacheview $x
# or in a pipeline:
@list |> map cacheview |> p
```

caesar_shift

`caesar_shift` — string (batch 2) builtin.

```
my $result = caesar_shift $x
# or in a pipeline:
@list |> map caesar_shift |> p
```

cagr

`cagr` — math / numeric (uncategorized batch) builtin.

```
my $result = cagr $x
# or in a pipeline:
@list |> map cagr |> p
```

cake_number

`cake_number` — batch 10 builtin.

```
my $result = cake_number $x
# or in a pipeline:
@list |> map cake_number |> p
```

cal_to_joules

`cal_to_joules` — file stat / path builtin.

```
my $result = cal_to_joules $input
```

calmar_ratio

`calmar_ratio` — batches 17-18 builtin.

```
my $result = calmar_ratio $x
# or in a pipeline:
@list |> map calmar_ratio |> p
```

calorimeter_dt

`calorimeter_dt` — batch 21 builtin.

```
my $result = calorimeter_dt $x
# or in a pipeline:
@list |> map calorimeter_dt |> p
```

calorimetric_heat_battery

`calorimetric_heat_battery` — batch 38 builtin.

```
my $result = calorimetric_heat_battery $x
# or in a pipeline:
@list |> map calorimetric_heat_battery |> p
```

camel_to_snake

`camel_to_snake` — string processing (uncategorized batch) builtin.

```
my $result = camel_to_snake $input
```

camel_words

`camel_words` — string helpers (batch 4) builtin.

```
my $result = camel_words $x
# or in a pipeline:
@list |> map camel_words |> p
```

canny_threshold_step

`canny_threshold_step` — batch 40 builtin.

```
my $result = canny_threshold_step $x
# or in a pipeline:
@list |> map canny_threshold_step |> p
```

cap

`cap` — trivial string ops builtin. Alias for `capitalize`.

```
my $result = cap $x
# or in a pipeline:
@list |> map cap |> p
```

capacitance

`capacitance` — physics formulas builtin.

```
my $result = capacitance $x
# or in a pipeline:
@list |> map capacitance |> p
```

capacitance_parallel

`capacitance_parallel` — batch 12 builtin.

```
my $result = capacitance_parallel $x
# or in a pipeline:
@list |> map capacitance_parallel |> p
```

capacitance_parallel_sum

`capacitance_parallel_sum` — batch 23 builtin.

```
my $result = capacitance_parallel_sum $x
# or in a pipeline:
@list |> map capacitance_parallel_sum |> p
```

capacitance_series

`capacitance_series` — batch 12 builtin.

```
my $result = capacitance_series $x
# or in a pipeline:
@list |> map capacitance_series |> p
```

capacitor_charge

`capacitor_charge` — batch 23 builtin.

```
my $result = capacitor_charge $x
# or in a pipeline:
@list |> map capacitor_charge |> p
```

capacitor_energy

`capacitor_energy` — physics formulas builtin.

```
my $result = capacitor_energy $x
# or in a pipeline:
@list |> map capacitor_energy |> p
```

capenergy

`capenergy` — physics formulas builtin. Alias for `capacitor_energy`.

```
my $result = capenergy $x
# or in a pipeline:
@list |> map capenergy |> p
```

capg

`capg` — extended stdlib builtin. Alias for `capture_groups`.

```
my $result = capg $x
# or in a pipeline:
@list |> map capg |> p
```

capitalize

`capitalize` — trivial string ops builtin.

```
my $result = capitalize $x
# or in a pipeline:
@list |> map capitalize |> p
```

capitalize_words

`capitalize_words` — batch 9 builtin.

```
my $result = capitalize_words $x
# or in a pipeline:
@list |> map capitalize_words |> p
```

capm

`capm($risk_free, $beta, $market_return)` — computes expected return using the Capital Asset Pricing Model: rf + β(rm - rf).

```
p capm(0.02, 1.2, 0.08)   # 9.2% expected return
```

capm_expected_return

`capm_expected_return` — batch 13 builtin.

```
my $result = capm_expected_return $x
# or in a pipeline:
@list |> map capm_expected_return |> p
```

capture_groups

`capture_groups` — extended stdlib builtin.

```
my $result = capture_groups $x
# or in a pipeline:
@list |> map capture_groups |> p
```

carmichael_q

`carmichael_q` — batch 10 builtin.

```
my $result = carmichael_q $x
# or in a pipeline:
@list |> map carmichael_q |> p
```

carnot_efficiency

`carnot_efficiency` — batch 12 builtin.

```
my $result = carnot_efficiency $x
# or in a pipeline:
@list |> map carnot_efficiency |> p
```

carrying_capacity_from_data

carrying_capacity_from_data — batch 22 builtin.

```
my $result = carrying_capacity_from_data $x
# or in a pipeline:
@list |> map carrying_capacity_from_data |> p
```

cart_n

cart_n — batch 10 builtin. Alias for cartesian_product_n.

```
my $result = cart_n $x
# or in a pipeline:
@list |> map cart_n |> p
```

cartan_determinant_a2

cartan_determinant_a2 — batch 37 builtin.

```
my $result = cartan_determinant_a2 $x
# or in a pipeline:
@list |> map cartan_determinant_a2 |> p
```

cartan_matrix_b2

cartan_matrix_b2 — batch 37 builtin.

```
my $result = cartan_matrix_b2 $x
# or in a pipeline:
@list |> map cartan_matrix_b2 |> p
```

cartesian_power

cartesian_power — extended stdlib batch 3 builtin.

```
my $result = cartesian_power $x
# or in a pipeline:
@list |> map cartesian_power |> p
```

cartesian_product

cartesian_product — python/ruby stdlib builtin.

```
my $result = cartesian_product $x
# or in a pipeline:
@list |> map cartesian_product |> p
```

cartesian_product_n

cartesian_product_n — batch 10 builtin.

```
my $result = cartesian_product_n $x
# or in a pipeline:
@list |> map cartesian_product_n |> p
```

cartpow

cartpow — extended stdlib batch 3 builtin. Alias for cartesian_power.

```
my $result = cartpow $x
# or in a pipeline:
@list |> map cartpow |> p
```

cas_admm_lasso_step

`cas_admm_lasso_step` — batch 44 builtin.

```
my $result = cas_admm_lasso_step $x
# or in a pipeline:
@list |> map cas_admm_lasso_step |> p
```

cas_alternating_projection

`cas_alternating_projection` — batch 44 builtin.

```
my $result = cas_alternating_projection $x
# or in a pipeline:
@list |> map cas_alternating_projection |> p
```

cas_arnoldi_iteration_step

`cas_arnoldi_iteration_step` — batch 44 builtin.

```
my $result = cas_arnoldi_iteration_step $x
# or in a pipeline:
@list |> map cas_arnoldi_iteration_step |> p
```

cas_basis_pursuit_step

`cas_basis_pursuit_step` — batch 44 builtin.

```
my $result = cas_basis_pursuit_step $x
# or in a pipeline:
@list |> map cas_basis_pursuit_step |> p
```

cas_block_lu_step

`cas_block_lu_step` — batch 44 builtin.

```
my $result = cas_block_lu_step $x
# or in a pipeline:
@list |> map cas_block_lu_step |> p
```

cas_buchberger_step

`cas_buchberger_step` — batch 44 builtin.

```
my $result = cas_buchberger_step $x
# or in a pipeline:
@list |> map cas_buchberger_step |> p
```

cas_bunch_kaufman_step

`cas_bunch_kaufman_step` — batch 44 builtin.

```
my $result = cas_bunch_kaufman_step $x
# or in a pipeline:
@list |> map cas_bunch_kaufman_step |> p
```

cas_burnside_count_step

`cas_burnside_count_step` — batch 44 builtin.

```
my $result = cas_burnside_count_step $x
# or in a pipeline:
@list |> map cas_burnside_count_step |> p
```

cas_chebyshev_eval

`cas_chebyshev_eval` — batch 44 builtin.

```
my $result = cas_chebyshev_eval $x
# or in a pipeline:
@list |> map cas_chebyshev_eval |> p
```

cas_cholesky_step

`cas_cholesky_step` — batch 44 builtin.

```
my $result = cas_cholesky_step $x
# or in a pipeline:
@list |> map cas_cholesky_step |> p
```

cas_classical_gram_schmidt

`cas_classical_gram_schmidt` — batch 44 builtin.

```
my $result = cas_classical_gram_schmidt $x
# or in a pipeline:
@list |> map cas_classical_gram_schmidt |> p
```

cas_companion_matrix_root

`cas_companion_matrix_root` — batch 44 builtin.

```
my $result = cas_companion_matrix_root $x
# or in a pipeline:
@list |> map cas_companion_matrix_root |> p
```

cas_constrained_ls_step

`cas_constrained_ls_step` — batch 44 builtin.

```
my $result = cas_constrained_ls_step $x
# or in a pipeline:
@list |> map cas_constrained_ls_step |> p
```

cas_continued_fraction_step

`cas_continued_fraction_step` — batch 44 builtin.

```
my $result = cas_continued_fraction_step $x
# or in a pipeline:
@list |> map cas_continued_fraction_step |> p
```

cas_cosamp_step

`cas_cosamp_step` — batch 44 builtin.

```
my $result = cas_cosamp_step $x
# or in a pipeline:
@list |> map cas_cosamp_step |> p
```

cas_descartes_rule_count

`cas_descartes_rule_count` — batch 44 builtin.

```
my $result = cas_descartes_rule_count $x
# or in a pipeline:
@list |> map cas_descartes_rule_count |> p
```

cas_drazin_inverse_step

`cas_drazin_inverse_step` — batch 44 builtin.

```
my $result = cas_drazin_inverse_step $x
# or in a pipeline:
@list |> map cas_drazin_inverse_step |> p
```

cas_dykstra_step

`cas_dykstra_step` — batch 44 builtin.

```
my $result = cas_dykstra_step $x
# or in a pipeline:
@list |> map cas_dykstra_step |> p
```

cas_eigenvalue_inverse_iteration

`cas_eigenvalue_inverse_iteration` — batch 44 builtin.

```
my $result = cas_eigenvalue_inverse_iteration $x
# or in a pipeline:
@list |> map cas_eigenvalue_inverse_iteration |> p
```

cas_elastic_net_step

`cas_elastic_net_step` — batch 44 builtin.

```
my $result = cas_elastic_net_step $x
# or in a pipeline:
@list |> map cas_elastic_net_step |> p
```

cas_expand_two_terms

`cas_expand_two_terms` — batch 44 builtin.

```
my $result = cas_expand_two_terms $x
# or in a pipeline:
@list |> map cas_expand_two_terms |> p
```

cas_extended_euclid_step

`cas_extended_euclid_step` — batch 44 builtin.

```
my $result = cas_extended_euclid_step $x
# or in a pipeline:
@list |> map cas_extended_euclid_step |> p
```

cas_factor_quadratic

`cas_factor_quadratic` — batch 44 builtin.

```
my $result = cas_factor_quadratic $x
# or in a pipeline:
@list |> map cas_factor_quadratic |> p
```

cas_gcd_polynomial_step

`cas_gcd_polynomial_step` — batch 44 builtin.

```
my $result = cas_gcd_polynomial_step $x
# or in a pipeline:
@list |> map cas_gcd_polynomial_step |> p
```

cas_gegenbauer_eval

`cas_gegenbauer_eval` — batch 44 builtin.

```
my $result = cas_gegenbauer_eval $x
# or in a pipeline:
@list |> map cas_gegenbauer_eval |> p
```

cas_generalized_eigen

`cas_generalized_eigen` — batch 44 builtin.

```
my $result = cas_generalized_eigen $x
# or in a pipeline:
@list |> map cas_generalized_eigen |> p
```

cas_givens_rotation_apply

`cas_givens_rotation_apply` — batch 44 builtin.

```
my $result = cas_givens_rotation_apply $x
# or in a pipeline:
@list |> map cas_givens_rotation_apply |> p
```

cas_groebner_lt_step

`cas_groebner_lt_step` — batch 44 builtin.

```
my $result = cas_groebner_lt_step $x
# or in a pipeline:
@list |> map cas_groebner_lt_step |> p
```

cas_hermite_eval

`cas_hermite_eval` — batch 44 builtin.

```
my $result = cas_hermite_eval $x
# or in a pipeline:
@list |> map cas_hermite_eval |> p
```

cas_hermite_normal_step

`cas_hermite_normal_step` — batch 44 builtin.

```
my $result = cas_hermite_normal_step $x
# or in a pipeline:
@list |> map cas_hermite_normal_step |> p
```

cas_householder_reflection

`cas_householder_reflection` — batch 44 builtin.

```
my $result = cas_householder_reflection $x
# or in a pipeline:
@list |> map cas_householder_reflection |> p
```

cas_iht_iteration

`cas_iht_iteration` — batch 44 builtin.

```
my $result = cas_iht_iteration $x
# or in a pipeline:
@list |> map cas_iht_iteration |> p
```

cas_indicator_simplex_proj

`cas_indicator_simplex_proj` — batch 44 builtin.

```
my $result = cas_indicator_simplex_proj $x
# or in a pipeline:
@list |> map cas_indicator_simplex_proj |> p
```

cas_jacobi_eigen_step

`cas_jacobi_eigen_step` — batch 44 builtin.

```
my $result = cas_jacobi_eigen_step $x
# or in a pipeline:
@list |> map cas_jacobi_eigen_step |> p
```

cas_jacobi_eval

`cas_jacobi_eval` — batch 44 builtin.

```
my $result = cas_jacobi_eval $x
# or in a pipeline:
@list |> map cas_jacobi_eval |> p
```

cas_kronecker_product_step

`cas_kronecker_product_step` — batch 44 builtin.

```
my $result = cas_kronecker_product_step $x
# or in a pipeline:
@list |> map cas_kronecker_product_step |> p
```

cas_lagrange_interpolate

`cas_lagrange_interpolate` — batch 44 builtin.

```
my $result = cas_lagrange_interpolate $x
# or in a pipeline:
@list |> map cas_lagrange_interpolate |> p
```

cas_laguerre_eval

`cas_laguerre_eval` — batch 44 builtin.

```
my $result = cas_laguerre_eval $x
# or in a pipeline:
@list |> map cas_laguerre_eval |> p
```

cas_lanczos_iteration_step

`cas_lanczos_iteration_step` — batch 44 builtin.

```
my $result = cas_lanczos_iteration_step $x
# or in a pipeline:
@list |> map cas_lanczos_iteration_step |> p
```

cas_lasso_soft_threshold

`cas_lasso_soft_threshold` — batch 44 builtin.

```
my $result = cas_lasso_soft_threshold $x
# or in a pipeline:
@list |> map cas_lasso_soft_threshold |> p
```

cas_ldlt_step

`cas_ldlt_step` — batch 44 builtin.

```
my $result = cas_ldlt_step $x
# or in a pipeline:
@list |> map cas_ldlt_step |> p
```

cas_least_squares_solve

`cas_least_squares_solve` — batch 44 builtin.

```
my $result = cas_least_squares_solve $x
# or in a pipeline:
@list |> map cas_least_squares_solve |> p
```

cas_legendre_eval

`cas_legendre_eval` — batch 44 builtin.

```
my $result = cas_legendre_eval $x
# or in a pipeline:
@list |> map cas_legendre_eval |> p
```

cas_lyapunov_continuous_step

`cas_lyapunov_continuous_step` — batch 44 builtin.

```
my $result = cas_lyapunov_continuous_step $x
# or in a pipeline:
@list |> map cas_lyapunov_continuous_step |> p
```

cas_lyapunov_discrete_step

`cas_lyapunov_discrete_step` — batch 44 builtin.

```
my $result = cas_lyapunov_discrete_step $x
# or in a pipeline:
@list |> map cas_lyapunov_discrete_step |> p
```

cas_macaulay_matrix_step

`cas_macaulay_matrix_step` — batch 44 builtin.

```
my $result = cas_macaulay_matrix_step $x
# or in a pipeline:
@list |> map cas_macaulay_matrix_step |> p
```

cas_matrix_exp_pade

`cas_matrix_exp_pade` — batch 44 builtin.

```
my $result = cas_matrix_exp_pade $x
# or in a pipeline:
@list |> map cas_matrix_exp_pade |> p
```

cas_matrix_function_step

`cas_matrix_function_step` — batch 44 builtin.

```
my $result = cas_matrix_function_step $x
# or in a pipeline:
@list |> map cas_matrix_function_step |> p
```

cas_matrix_log_step

`cas_matrix_log_step` — batch 44 builtin.

```
my $result = cas_matrix_log_step $x
# or in a pipeline:
@list |> map cas_matrix_log_step |> p
```

cas_matrix_pencil_step

`cas_matrix_pencil_step` — batch 44 builtin.

```
my $result = cas_matrix_pencil_step $x
# or in a pipeline:
@list |> map cas_matrix_pencil_step |> p
```

cas_matrix_sqrt_step

`cas_matrix_sqrt_step` — batch 44 builtin.

```
my $result = cas_matrix_sqrt_step $x
# or in a pipeline:
@list |> map cas_matrix_sqrt_step |> p
```

cas_minimal_polynomial

`cas_minimal_polynomial` — batch 44 builtin.

```
my $result = cas_minimal_polynomial $x
# or in a pipeline:
@list |> map cas_minimal_polynomial |> p
```

cas_modified_cholesky

`cas_modified_cholesky` — batch 44 builtin.

```
my $result = cas_modified_cholesky $x
# or in a pipeline:
@list |> map cas_modified_cholesky |> p
```

cas_modified_gram_schmidt

`cas_modified_gram_schmidt` — batch 44 builtin.

```
my $result = cas_modified_gram_schmidt $x
# or in a pipeline:
@list |> map cas_modified_gram_schmidt |> p
```

cas_modular_inverse

`cas_modular_inverse` — batch 44 builtin.

```
my $result = cas_modular_inverse $x
# or in a pipeline:
@list |> map cas_modular_inverse |> p
```

cas_moore_penrose_step

`cas_moore_penrose_step` — batch 44 builtin.

```
my $result = cas_moore_penrose_step $x
# or in a pipeline:
@list |> map cas_moore_penrose_step |> p
```

cas_omp_step

`cas_omp_step` — batch 44 builtin.

```
my $result = cas_omp_step $x
# or in a pipeline:
@list |> map cas_omp_step |> p
```

cas_partial_fraction_simple

`cas_partial_fraction_simple` — batch 44 builtin.

```
my $result = cas_partial_fraction_simple $x
# or in a pipeline:
@list |> map cas_partial_fraction_simple |> p
```

cas_pivoted_lu_step

`cas_pivoted_lu_step` — batch 44 builtin.

```
my $result = cas_pivoted_lu_step $x
# or in a pipeline:
@list |> map cas_pivoted_lu_step |> p
```

cas_polar_decomposition

`cas_polar_decomposition` — batch 44 builtin.

```
my $result = cas_polar_decomposition $x
# or in a pipeline:
@list |> map cas_polar_decomposition |> p
```

cas_polya_enumeration_step

`cas_polya_enumeration_step` — batch 44 builtin.

```
my $result = cas_polya_enumeration_step $x
# or in a pipeline:
@list |> map cas_polya_enumeration_step |> p
```

cas_polynomial_div_step

`cas_polynomial_div_step` — batch 44 builtin.

```
my $result = cas_polynomial_div_step $x
# or in a pipeline:
@list |> map cas_polynomial_div_step |> p
```

cas_polynomial_gcd_step

`cas_polynomial_gcd_step` — batch 44 builtin.

```
my $result = cas_polynomial_gcd_step $x
# or in a pipeline:
@list |> map cas_polynomial_gcd_step |> p
```

cas_polynomial_roots_kahan

`cas_polynomial_roots_kahan` — batch 44 builtin.

```
my $result = cas_polynomial_roots_kahan $x
# or in a pipeline:
@list |> map cas_polynomial_roots_kahan |> p
```

cas_proj_box

`cas_proj_box` — batch 44 builtin.

```
my $result = cas_proj_box $x
# or in a pipeline:
@list |> map cas_proj_box |> p
```

cas_proj_exp_cone

`cas_proj_exp_cone` — batch 44 builtin.

```
my $result = cas_proj_exp_cone $x
# or in a pipeline:
@list |> map cas_proj_exp_cone |> p
```

cas_proj_l1_ball

`cas_proj_l1_ball` — batch 44 builtin.

```
my $result = cas_proj_l1_ball $x
# or in a pipeline:
@list |> map cas_proj_l1_ball |> p
```

cas_proj_l2_ball

`cas_proj_l2_ball` — batch 44 builtin.

```
my $result = cas_proj_l2_ball $x
# or in a pipeline:
@list |> map cas_proj_l2_ball |> p
```

cas_proj_psd_cone

`cas_proj_psd_cone` — batch 44 builtin.

```
my $result = cas_proj_psd_cone $x
# or in a pipeline:
@list |> map cas_proj_psd_cone |> p
```

cas_proj_soc_step

`cas_proj_soc_step` — batch 44 builtin.

```
my $result = cas_proj_soc_step $x
# or in a pipeline:
@list |> map cas_proj_soc_step |> p
```

cas_proximal_l1_step

`cas_proximal_l1_step` — batch 44 builtin.

```
my $result = cas_proximal_l1_step $x
# or in a pipeline:
@list |> map cas_proximal_l1_step |> p
```

cas_proximal_l2_step

`cas_proximal_l2_step` — batch 44 builtin.

```
my $result = cas_proximal_l2_step $x
# or in a pipeline:
@list |> map cas_proximal_l2_step |> p
```

cas_proximal_l_inf_step

`cas_proximal_l_inf_step` — batch 44 builtin.

```
my $result = cas_proximal_l_inf_step $x
# or in a pipeline:
@list |> map cas_proximal_l_inf_step |> p
```

cas_pseudoinverse_step

`cas_pseudoinverse_step` — batch 44 builtin.

```
my $result = cas_pseudoinverse_step $x
# or in a pipeline:
@list |> map cas_pseudoinverse_step |> p
```

cas_qr_iteration_step

`cas_qr_iteration_step` — batch 44 builtin.

```
my $result = cas_qr_iteration_step $x
# or in a pipeline:
@list |> map cas_qr_iteration_step |> p
```

cas_quasi_triangular

`cas_quasi_triangular` — batch 44 builtin.

```
my $result = cas_quasi_triangular $x
# or in a pipeline:
@list |> map cas_quasi_triangular |> p
```

cas_radical_simplify

`cas_radical_simplify` — batch 44 builtin.

```
my $result = cas_radical_simplify $x
# or in a pipeline:
@list |> map cas_radical_simplify |> p
```

cas_rank_revealing_qr

`cas_rank_revealing_qr` — batch 44 builtin.

```
my $result = cas_rank_revealing_qr $x
# or in a pipeline:
@list |> map cas_rank_revealing_qr |> p
```

cas_regularized_lsq_tikhonov

`cas_regularized_lsq_tikhonov` — batch 44 builtin.

```
my $result = cas_regularized_lsq_tikhonov $x
# or in a pipeline:
@list |> map cas_regularized_lsq_tikhonov |> p
```

cas_resultant_two

cas_resultant_two — batch 44 builtin.

```
my $result = cas_resultant_two $x
# or in a pipeline:
@list |> map cas_resultant_two |> p
```

cas_resultant_x_y

cas_resultant_x_y — batch 44 builtin.

```
my $result = cas_resultant_x_y $x
# or in a pipeline:
@list |> map cas_resultant_x_y |> p
```

cas_riccati_continuous_step

cas_riccati_continuous_step — batch 44 builtin.

```
my $result = cas_riccati_continuous_step $x
# or in a pipeline:
@list |> map cas_riccati_continuous_step |> p
```

cas_riccati_discrete_step

cas_riccati_discrete_step — batch 44 builtin.

```
my $result = cas_riccati_discrete_step $x
# or in a pipeline:
@list |> map cas_riccati_discrete_step |> p
```

cas_root_isolate_step

cas_root_isolate_step — batch 44 builtin.

```
my $result = cas_root_isolate_step $x
# or in a pipeline:
@list |> map cas_root_isolate_step |> p
```

cas_schur_decomposition_step

cas_schur_decomposition_step — batch 44 builtin.

```
my $result = cas_schur_decomposition_step $x
# or in a pipeline:
@list |> map cas_schur_decomposition_step |> p
```

cas_simplify_term

cas_simplify_term — batch 44 builtin.

```
my $result = cas_simplify_term $x
# or in a pipeline:
@list |> map cas_simplify_term |> p
```

cas_singular_value_step

cas_singular_value_step — batch 44 builtin.

```
my $result = cas_singular_value_step $x
# or in a pipeline:
@list |> map cas_singular_value_step |> p
```

cas_smith_normal_step

`cas_smith_normal_step` — batch 44 builtin.

```
my $result = cas_smith_normal_step $x
# or in a pipeline:
@list |> map cas_smith_normal_step |> p
```

cas_solve_cubic

`cas_solve_cubic` — batch 44 builtin.

```
my $result = cas_solve_cubic $x
# or in a pipeline:
@list |> map cas_solve_cubic |> p
```

cas_solve_linear

`cas_solve_linear` — batch 44 builtin.

```
my $result = cas_solve_linear $x
# or in a pipeline:
@list |> map cas_solve_linear |> p
```

cas_solve_polynomial_n

`cas_solve_polynomial_n` — batch 44 builtin.

```
my $result = cas_solve_polynomial_n $x
# or in a pipeline:
@list |> map cas_solve_polynomial_n |> p
```

cas_solve_quadratic

`cas_solve_quadratic` — batch 44 builtin.

```
my $result = cas_solve_quadratic $x
# or in a pipeline:
@list |> map cas_solve_quadratic |> p
```

cas_solve_quartic

`cas_solve_quartic` — batch 44 builtin.

```
my $result = cas_solve_quartic $x
# or in a pipeline:
@list |> map cas_solve_quartic |> p
```

cas_sturm_sequence_step

`cas_sturm_sequence_step` — batch 44 builtin.

```
my $result = cas_sturm_sequence_step $x
# or in a pipeline:
@list |> map cas_sturm_sequence_step |> p
```

cas_subresultant_two

`cas_subresultant_two` — batch 44 builtin.

```
my $result = cas_subresultant_two $x
# or in a pipeline:
@list |> map cas_subresultant_two |> p
```

cas_sylvester_equation_step

`cas_sylvester_equation_step` — batch 44 builtin.

```
my $result = cas_sylvester_equation_step $x
# or in a pipeline:
@list |> map cas_sylvester_equation_step |> p
```

cas_taylor_coefficient

`cas_taylor_coefficient` — batch 44 builtin.

```
my $result = cas_taylor_coefficient $x
# or in a pipeline:
@list |> map cas_taylor_coefficient |> p
```

cas_total_least_squares

`cas_total_least_squares` — batch 44 builtin.

```
my $result = cas_total_least_squares $x
# or in a pipeline:
@list |> map cas_total_least_squares |> p
```

cas_truncated_lsq

`cas_truncated_lsq` — batch 44 builtin.

```
my $result = cas_truncated_lsq $x
# or in a pipeline:
@list |> map cas_truncated_lsq |> p
```

cas_truncated_svd_value

`cas_truncated_svd_value` — batch 44 builtin.

```
my $result = cas_truncated_svd_value $x
# or in a pipeline:
@list |> map cas_truncated_svd_value |> p
```

cas_vec_operator_step

`cas_vec_operator_step` — batch 44 builtin.

```
my $result = cas_vec_operator_step $x
# or in a pipeline:
@list |> map cas_vec_operator_step |> p
```

cas_woodbury_identity

`cas_woodbury_identity` — batch 44 builtin.

```
my $result = cas_woodbury_identity $x
# or in a pipeline:
@list |> map cas_woodbury_identity |> p
```

casimir_eigenvalue_su2

`casimir_eigenvalue_su2` — batch 37 builtin.

```
my $result = casimir_eigenvalue_su2 $x
# or in a pipeline:
@list |> map casimir_eigenvalue_su2 |> p
```

catalan

`catalan_constant` (alias `catalan`) — Catalan's constant G ≈ 0.9159. Appears in combinatorics and series.

```
p catalan   # 0.9159655941772190
```

catn

`catn` — extended stdlib batch 3 builtin. Alias for `catalan`.

```
my $result = catn $x
# or in a pipeline:
@list |> map catn |> p
```

cauchy_pdf

`cauchy_pdf` (alias `cauchypdf`) evaluates the Cauchy distribution PDF at x with location x0 and scale gamma.

```
p cauchypdf(0, 0, 1)  # peak of standard Cauchy
```

caverphone_2

`caverphone_2` — batch 69 builtin.

```
my $result = caverphone_2 $x
# or in a pipeline:
@list |> map caverphone_2 |> p
```

cbind

`cbind` — bind matrices by columns (horizontal join). Like R's cbind().

```
my $m = cbind([[1],[2]], [[3],[4]])  # [[1,3],[2,4]]
```

cblend

`cblend` — color operations builtin. Alias for `color_blend`.

```
my $result = cblend $x
# or in a pipeline:
@list |> map cblend |> p
```

cbor_encode_str

`cbor_encode_str` — batch 62 builtin.

```
my $result = cbor_encode_str $x
# or in a pipeline:
@list |> map cbor_encode_str |> p
```

cbor_encode_uint

`cbor_encode_uint` — batch 62 builtin.

```
my $result = cbor_encode_uint $x
# or in a pipeline:
@list |> map cbor_encode_uint |> p
```

cbrt

`cbrt` — trivial numeric / predicate builtins builtin.

```
my $result = cbrt $x
# or in a pipeline:
@list |> map cbrt |> p
```

cc_count

`cc_count` — batch 24 builtin.

```
my $result = cc_count $x
# or in a pipeline:
@list |> map cc_count |> p
```

cc_labels

`cc_labels` — batch 24 builtin.

```
my $result = cc_labels $x
# or in a pipeline:
@list |> map cc_labels |> p
```

ccgraph

`ccgraph` — extended stdlib builtin. Alias for `connected_components_graph`.

```
my $result = ccgraph $x
# or in a pipeline:
@list |> map ccgraph |> p
```

ccompl

`ccompl` — color operations builtin. Alias for `color_complement`.

```
my $result = ccompl $x
# or in a pipeline:
@list |> map ccompl |> p
```

ccount

`ccount` — extended stdlib batch 3 builtin. Alias for `count_chars`.

```
my $result = ccount $x
# or in a pipeline:
@list |> map ccount |> p
```

ccx_gate

`ccx_gate` — batches 14-16 builtin. Alias for `toffoli_gate`.

```
my $result = ccx_gate $x
# or in a pipeline:
@list |> map ccx_gate |> p
```

cdarken

`cdarken` — color operations builtin. Alias for `color_darken`.

```
my $result = cdarken $x
# or in a pipeline:
@list |> map cdarken |> p
```

cdist

`cdist` — color operations builtin. Alias for `color_distance`.

```
my $result = cdist $x
# or in a pipeline:
@list |> map cdist |> p
```

cds_upfront

`cds_upfront` — batch 20 builtin.

```
my $result = cds_upfront $x
# or in a pipeline:
@list |> map cds_upfront |> p
```

cech_zero_cohomology

`cech_zero_cohomology` — batch 37 builtin.

```
my $result = cech_zero_cohomology $x
# or in a pipeline:
@list |> map cech_zero_cohomology |> p
```

ceil

`ceil` — trivial numeric / predicate builtins builtin.

```
my $result = ceil $x
# or in a pipeline:
@list |> map ceil |> p
```

ceil_div

`ceil_div` — trig / math (batch 2) builtin.

```
my $result = ceil_div $x
# or in a pipeline:
@list |> map ceil_div |> p
```

ceil_each

`ceil_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = ceil_each $x
# or in a pipeline:
@list |> map ceil_each |> p
```

ceiling

`ceiling` — trivial numeric / predicate builtins builtin. Alias for `ceil`.

```
my $result = ceiling $x
# or in a pipeline:
@list |> map ceiling |> p
```

center

`center` — trivial string ops builtin.

```
my $result = center $x
# or in a pipeline:
@list |> map center |> p
```

center_text

`center_text` — extended stdlib builtin.

```
my $result = center_text $x
# or in a pipeline:
@list |> map center_text |> p
```

centered_hexagonal_number

`centered_hexagonal_number` — batch 10 builtin.

```
my $result = centered_hexagonal_number $x
# or in a pipeline:
@list |> map centered_hexagonal_number |> p
```

centered_pentagonal

`centered_pentagonal` — batch 11 builtin.

```
my $result = centered_pentagonal $x
# or in a pipeline:
@list |> map centered_pentagonal |> p
```

centered_square

`centered_square` — batch 11 builtin.

```
my $result = centered_square $x
# or in a pipeline:
@list |> map centered_square |> p
```

centered_triangular

`centered_triangular` — batch 11 builtin.

```
my $result = centered_triangular $x
# or in a pipeline:
@list |> map centered_triangular |> p
```

central_polygonal

`central_polygonal` — batch 11 builtin.

```
my $result = central_polygonal $x
# or in a pipeline:
@list |> map central_polygonal |> p
```

centrality_betweenness

`centrality_betweenness` — batch 75 builtin.

```
my $result = centrality_betweenness $x
# or in a pipeline:
@list |> map centrality_betweenness |> p
```

centrality_closeness

`centrality_closeness` — batch 75 builtin.

```
my $result = centrality_closeness $x
# or in a pipeline:
@list |> map centrality_closeness |> p
```

centrality_degree

`centrality_degree` — batch 75 builtin.

```
my $result = centrality_degree $x
# or in a pipeline:
@list |> map centrality_degree |> p
```

centrality_eigenvector

`centrality_eigenvector` — batch 75 builtin.

```
my $result = centrality_eigenvector $x
# or in a pipeline:
@list |> map centrality_eigenvector |> p
```

centrality_katz

`centrality_katz` — batch 75 builtin.

```
my $result = centrality_katz $x
# or in a pipeline:
@list |> map centrality_katz |> p
```

centrip

`centripetal_force($mass, $velocity, $radius)` (alias `centrip`) — F = mv²/r for circular motion.

```
p centrip(1000, 20, 50)   # 8000 N (car turning)
```

centripetal_accel

`centripetal_accel` — batch 12 builtin.

```
my $result = centripetal_accel $x
# or in a pipeline:
@list |> map centripetal_accel |> p
```

centroid

`centroid(@points)` — computes the centroid (geometric center) of 2D points. Points as flat list [x1,y1,x2,y2,...]. Returns [$cx, $cy].

```
my @triangle = (0,0, 3,0, 0,3)
my $c = centroid(@triangle)
p @$c   # (1, 1)
```

centroid_nd

`centroid_nd` — batch 28 builtin.

```
my $result = centroid_nd $x
# or in a pipeline:
@list |> map centroid_nd |> p
```

cents_between

`cents_between` — batch 30 builtin.

```
my $result = cents_between $x
# or in a pipeline:
@list |> map cents_between |> p
```

cents_between_freqs

`cents_between_freqs` — batch 64 builtin.

```
my $result = cents_between_freqs $x
# or in a pipeline:
@list |> map cents_between_freqs |> p
```

ces_production

`ces_production` — batch 73 builtin.

```
my $result = ces_production $x
# or in a pipeline:
@list |> map ces_production |> p
```

cfl_number

`cfl_number` — batch 34 builtin.

```
my $result = cfl_number $x
# or in a pipeline:
@list |> map cfl_number |> p
```

cfr_case_fatality

`cfr_case_fatality` — batch 61 builtin.

```
my $result = cfr_case_fatality $x
# or in a pipeline:
@list |> map cfr_case_fatality |> p
```

cg_beta_fr

`cg_beta_fr` — batch 19 builtin.

```
my $result = cg_beta_fr $x
# or in a pipeline:
@list |> map cg_beta_fr |> p
```

cg_beta_pr

`cg_beta_pr` — batch 19 builtin.

```
my $result = cg_beta_pr $x
# or in a pipeline:
@list |> map cg_beta_pr |> p
```

cg_simple

`cg_simple` — batch 32 builtin.

```
my $result = cg_simple $x
# or in a pipeline:
@list |> map cg_simple |> p
```

cgemm

`cgemm` — batch 66 builtin.

```
my $result = cgemm $x
# or in a pipeline:
@list |> map cgemm |> p
```

cgray

`cgray` — color operations builtin. Alias for `color_grayscale`.

```
my $result = cgray $x
# or in a pipeline:
@list |> map cgray |> p
```

chacha20_qround

`chacha20_qround` — batches 14-16 builtin.

```
my $result = chacha20_qround $x
# or in a pipeline:
@list |> map chacha20_qround |> p
```

chain_from

`chain_from` — python/ruby stdlib builtin.

```
my $result = chain_from $x
# or in a pipeline:
@list |> map chain_from |> p
```

chain_ladder_step

`chain_ladder_step` — batch 60 builtin.

```
my $result = chain_ladder_step $x
# or in a pipeline:
@list |> map chain_ladder_step |> p
```

chain_rule_entropy

`chain_rule_entropy` — batch 40 builtin.

```
my $result = chain_rule_entropy $x
# or in a pipeline:
@list |> map chain_rule_entropy |> p
```

chandrasekhar_mass

`chandrasekhar_mass` — batch 12 builtin.

```
my $result = chandrasekhar_mass $x
# or in a pipeline:
@list |> map chandrasekhar_mass |> p
```

chapman_estimator

`chapman_estimator` — batch 22 builtin.

```
my $result = chapman_estimator $x
# or in a pipeline:
@list |> map chapman_estimator |> p
```

char_at

`char_at` — extended stdlib builtin.

```
my $result = char_at $x
# or in a pipeline:
@list |> map char_at |> p
```

char_count

`char_count` — trivial string ops builtin.

```
my $result = char_count $x
# or in a pipeline:
@list |> map char_count |> p
```

char_devices

`char_devices` — filesystem extensions builtin.

```
my $result = char_devices $x
# or in a pipeline:
@list |> map char_devices |> p
```

char_frequencies

`char_frequencies` — string processing (uncategorized batch) builtin.

```
my $result = char_frequencies $x
# or in a pipeline:
@list |> map char_frequencies |> p
```

char_length

`char_length` — string helpers (batch 4) builtin.

```
my $result = char_length $x
# or in a pipeline:
@list |> map char_length |> p
```

character_su2

`character_su2` — batch 37 builtin.

```
my $result = character_su2 $x
# or in a pipeline:
@list |> map character_su2 |> p
```

character_sun

`character_sun` — batch 37 builtin.

```
my $result = character_sun $x
# or in a pipeline:
@list |> map character_sun |> p
```

characteristic_function

`characteristic_function` — batch 43 builtin.

```
my $result = characteristic_function $x
# or in a pipeline:
@list |> map characteristic_function |> p
```

charcodes_to_string

`charcodes_to_string` — batch 13 builtin.

```
my $result = charcodes_to_string $input
```

charfreq

`charfreq` — string processing (uncategorized batch) builtin. Alias for `char_frequencies`.

```
my $result = charfreq $x
# or in a pipeline:
@list |> map charfreq |> p
```

charge_capacity_battery

`charge_capacity_battery` — batch 38 builtin.

```
my $result = charge_capacity_battery $x
# or in a pipeline:
@list |> map charge_capacity_battery |> p
```

charge_transfer_resistance

`charge_transfer_resistance` — batch 38 builtin.

```
my $result = charge_transfer_resistance $x
# or in a pipeline:
@list |> map charge_transfer_resistance |> p
```

chars_to_string

`chars_to_string` — string helpers (batch 4) builtin.

```
my $result = chars_to_string $input
```

chc_bound_makespan

`chc_bound_makespan` — batch 41 builtin.

```
my $result = chc_bound_makespan $x
# or in a pipeline:
@list |> map chc_bound_makespan |> p
```

cheapest_insertion_step

`cheapest_insertion_step` — batch 41 builtin.

```
my $result = cheapest_insertion_step $x
# or in a pipeline:
@list |> map cheapest_insertion_step |> p
```

cheby1_lp

`cheby1_lp` — batch 74 builtin.

```
my $result = cheby1_lp $x
# or in a pipeline:
@list |> map cheby1_lp |> p
```

cheby2_lp

`cheby2_lp` — batch 74 builtin.

```
my $result = cheby2_lp $x
# or in a pipeline:
@list |> map cheby2_lp |> p
```

chebyshev_distance

`chebyshev_distance` — math functions builtin.

```
my $result = chebyshev_distance $x
# or in a pipeline:
@list |> map chebyshev_distance |> p
```

chebyt

`chebyt` — batch 72 builtin.

```
my $result = chebyt $x
# or in a pipeline:
@list |> map chebyt |> p
```

chebyu

`chebyu` — batch 72 builtin.

```
my $result = chebyu $x
# or in a pipeline:
@list |> map chebyu |> p
```

chern_first_2d

`chern_first_2d` — batch 37 builtin.

```
my $result = chern_first_2d $x
# or in a pipeline:
@list |> map chern_first_2d |> p
```

chern_pontryagin_4d_step

`chern_pontryagin_4d_step` — batch 39 builtin.

```
my $result = chern_pontryagin_4d_step $x
# or in a pipeline:
@list |> map chern_pontryagin_4d_step |> p
```

chern_simons_action

`chern_simons_action` — batch 37 builtin.

```
my $result = chern_simons_action $x
# or in a pipeline:
@list |> map chern_simons_action |> p
```

chezy_velocity

`chezy_velocity` — batch 13 builtin.

```
my $result = chezy_velocity $x
# or in a pipeline:
@list |> map chezy_velocity |> p
```

chfr

`chfr` — python/ruby stdlib builtin. Alias for `chain_from`.

```
my $result = chfr $x
# or in a pipeline:
@list |> map chfr |> p
```

chi2

`chi_square_stat(\@observed, \@expected)` (alias `chi2`) — computes the chi-squared test statistic comparing observed vs expected frequencies. Useful for goodness-of-fit tests. Returns the chi-squared value.

```
my @obs = (10, 20, 30)
my @exp = (15, 20, 25)
p chi2(\@obs, \@exp)   # chi-squared statistic
```

chi2_pdf

`chi2_pdf` (alias `chi2pdf`) evaluates the chi-squared distribution PDF at x with k degrees of freedom.

```
p chi2pdf(3.84, 1)  # p-value boundary for df=1
```

chicken_game_payoff

`chicken_game_payoff` — batch 43 builtin.

```
my $result = chicken_game_payoff $x
# or in a pipeline:
@list |> map chicken_game_payoff |> p
```

chinese_lunation_winter

`chinese_lunation_winter` — batch 52 builtin.

```
my $result = chinese_lunation_winter $x
# or in a pipeline:
@list |> map chinese_lunation_winter |> p
```

chinese_new_year

`chinese_new_year` — batch 82 builtin.

```
my $result = chinese_new_year $x
# or in a pipeline:
@list |> map chinese_new_year |> p
```

chinese_remainder

`chinese_remainder` (alias `crt`) solves a system of simultaneous congruences via the Chinese Remainder Theorem.

```
p crt([2,3,2], [3,5,7])  # 23 (x≡2 mod 3, x≡3 mod 5, x≡2 mod 7)
```

chinese_year_zodiac

`chinese_year_zodiac` — batch 52 builtin.

```
my $result = chinese_year_zodiac $x
# or in a pipeline:
@list |> map chinese_year_zodiac |> p
```

chinese_zodiac

`chinese_zodiac` — batch 9 builtin.

```
my $result = chinese_zodiac $x
# or in a pipeline:
@list |> map chinese_zodiac |> p
```

chirp

`chirp` generates a linear chirp signal sweeping from f0 to f1 Hz. Args: n_samples, f0, f1, sample_rate.

```
my @sig = @{chirp(1000, 100, 1000, 8000)}
```

chirp_linear

`chirp_linear` — batch 25 builtin.

```
my $result = chirp_linear $x
# or in a pipeline:
@list |> map chirp_linear |> p
```

chirp_mass

`chirp_mass` — batch 31 builtin.

```
my $result = chirp_mass $x
# or in a pipeline:
@list |> map chirp_mass |> p
```

chirp_mass_inspiral_step

`chirp_mass_inspiral_step` — batch 39 builtin.

```
my $result = chirp_mass_inspiral_step $x
# or in a pipeline:
@list |> map chirp_mass_inspiral_step |> p
```

chisquare_metric

`chisquare_metric` — batch 40 builtin.

```
my $result = chisquare_metric $x
# or in a pipeline:
@list |> map chisquare_metric |> p
```

chkstr

`chkstr` — string processing (uncategorized batch) builtin. Alias for `chunk_string`.

```
my $result = chkstr $x
# or in a pipeline:
@list |> map chkstr |> p
```

chkw

`chkw` — ruby enumerable extras builtin. Alias for `chunk_while`.

```
my $result = chkw $x
# or in a pipeline:
@list |> map chkw |> p
```

chlorine_radical_decay

`chlorine_radical_decay` — batch 42 builtin.

```
my $result = chlorine_radical_decay $x
# or in a pipeline:
@list |> map chlorine_radical_decay |> p
```

cholesky

`matrix_cholesky` (aliases `mchol`, `cholesky`) computes the Cholesky decomposition of a symmetric positive-definite matrix. Returns lower-triangular L where M = L·L^T.

```
my $L = cholesky([[4,2],[2,3]])
```

chomp_str

`chomp_str` — string helpers (batch 4) builtin.

```
my $result = chomp_str $x
# or in a pipeline:
@list |> map chomp_str |> p
```

choose

`choose` — batch 57 builtin.

```
my $result = choose $x
# or in a pipeline:
@list |> map choose |> p
```

chop_str

`chop_str` — string helpers (batch 4) builtin.

```
my $result = chop_str $x
# or in a pipeline:
@list |> map chop_str |> p
```

chord_quality_classify

`chord_quality_classify` — batch 64 builtin.

```
my $result = chord_quality_classify $x
# or in a pipeline:
@list |> map chord_quality_classify |> p
```

chord_root_inversion

chord_root_inversion — batch 64 builtin.

```
my $result = chord_root_inversion $x
# or in a pipeline:
@list |> map chord_root_inversion |> p
```

chord_to_freqs

chord_to_freqs — batch 9 builtin.

```
my $result = chord_to_freqs $input
```

chord_voicing_close

chord_voicing_close — batch 64 builtin.

```
my $result = chord_voicing_close $x
# or in a pipeline:
@list |> map chord_voicing_close |> p
```

chow_test_stat

chow_test_stat — batch 36 builtin.

```
my $result = chow_test_stat $x
# or in a pipeline:
@list |> map chow_test_stat |> p
```

christoffel_first_kind_step

christoffel_first_kind_step — batch 39 builtin.

```
my $result = christoffel_first_kind_step $x
# or in a pipeline:
@list |> map christoffel_first_kind_step |> p
```

christoffel_second_kind_step

christoffel_second_kind_step — batch 39 builtin.

```
my $result = christoffel_second_kind_step $x
# or in a pipeline:
@list |> map christoffel_second_kind_step |> p
```

christoffel_symbol_normalize

christoffel_symbol_normalize — batch 39 builtin.

```
my $result = christoffel_symbol_normalize $x
# or in a pipeline:
@list |> map christoffel_symbol_normalize |> p
```

christoffel_word

christoffel_word — batches 14-16 builtin.

```
my $result = christoffel_word $x
# or in a pipeline:
@list |> map christoffel_word |> p
```

christofides_ratio_bound

`christofides_ratio_bound` — batch 41 builtin.

```
my $result = christofides_ratio_bound $x
# or in a pipeline:
@list |> map christofides_ratio_bound |> p
```

chroma_feature_step

`chroma_feature_step` — batch 40 builtin.

```
my $result = chroma_feature_step $x
# or in a pipeline:
@list |> map chroma_feature_step |> p
```

chromatic_homology_rank

`chromatic_homology_rank` — batch 37 builtin.

```
my $result = chromatic_homology_rank $x
# or in a pipeline:
@list |> map chromatic_homology_rank |> p
```

chromatic_number_greedy

`chromatic_number_greedy` — batch 24 builtin.

```
my $result = chromatic_number_greedy $x
# or in a pipeline:
@list |> map chromatic_number_greedy |> p
```

chroot

`chroot` — process / system builtin.

```
my $result = chroot $x
# or in a pipeline:
@list |> map chroot |> p
```

chsh_expectation

`chsh_expectation` — batch 32 builtin.

```
my $result = chsh_expectation $x
# or in a pipeline:
@list |> map chsh_expectation |> p
```

chull

`convex_hull(@points)` (alias `chull`) — computes the convex hull of a set of 2D points. Points given as flat list [x1,y1,x2,y2,...]. Returns the hull vertices in counterclockwise order. Uses Graham scan algorithm.

```
my @pts = (0,0, 1,1, 2,0, 1,0.5)
my $hull = chull(@pts)
p @$hull
```

chunk_by

`chunk_by` — functional / iterator builtin.

```
my $result = chunk_by $x
# or in a pipeline:
@list |> map chunk_by |> p
```

chunk_n

`chunk_n` — collection (batch 2) builtin. Alias for `group_of_n`.

```
my $result = chunk_n $x
# or in a pipeline:
@list |> map chunk_n |> p
```

chunk_string

`chunk_string` — string processing (uncategorized batch) builtin.

```
my $result = chunk_string $x
# or in a pipeline:
@list |> map chunk_string |> p
```

chunk_while

`chunk_while` — ruby enumerable extras builtin.

```
my $result = chunk_while $x
# or in a pipeline:
@list |> map chunk_while |> p
```

church_numeral_n

`church_numeral_n` — batch 67 builtin.

```
my $result = church_numeral_n $x
# or in a pipeline:
@list |> map church_numeral_n |> p
```

chvatal_gomory_cut

`chvatal_gomory_cut` — batch 41 builtin.

```
my $result = chvatal_gomory_cut $x
# or in a pipeline:
@list |> map chvatal_gomory_cut |> p
```

ci

`confidence_interval` (alias `ci`) computes a confidence interval for the mean. Default 95%. Returns [lower, upper].

```
my ($lo, $hi) = @{ci([10,12,14,11,13])}
p "95%% CI: $lo to $hi"
my ($lo99, $hi99) = @{ci([10,12,14,11,13], 0.99)}
```

cindex

`cindex STRING, SUBSTRING [, FROM]` — codepoint-indexed equivalent of Perl 5's `index`. Returns the zero-based codepoint position of the first occurrence, or -1 on miss. FROM is also a codepoint position. Pairs with `[i]` / `[a:b]` slice operators and `len` so search positions and slice bounds share one coordinate system.

Use `cindex` whenever your code mixes string search with `[]` slicing — Perl 5 `index` returns byte positions and silently misaligns when fed into `[]` (which is codepoint-based). `index` stays byte-indexed because `length` and `substr` are, and changing it would break Perl-5 binary-protocol code; `cindex` is the stryke-extension counterpart.

```
my $s = "hello 🌍 world"
p index  $s, "world"        # 9 (bytes — `🌍` is 3 bytes)
p cindex $s, "world"        # 7 (codepoints)
my $i = cindex $s, "world"
p $s[$i : len($s) - 1]      # "world" — slice bounds line up
```

cinv

`cinv` — extended stdlib batch 3 builtin. Alias for `count_inversions`.

```
my $result = cinv $x
# or in a pipeline:
@list |> map cinv |> p
```

cinvert

`cinvert` — color operations builtin. Alias for `color_invert`.

```
my $result = cinvert $x
# or in a pipeline:
@list |> map cinvert |> p
```

cir_bond

`cir_bond` — batch 20 builtin.

```
my $result = cir_bond $x
# or in a pipeline:
@list |> map cir_bond |> p
```

circ3pt

`circ3pt` — geometry (extended) builtin. Alias for `circle_from_three_points`.

```
my $result = circ3pt $x
# or in a pipeline:
@list |> map circ3pt |> p
```

circle_3_points

`circle_3_points` — batches 17-18 builtin.

```
my $result = circle_3_points $x
# or in a pipeline:
@list |> map circle_3_points |> p
```

circle_from_three_points

`circle_from_three_points($x1, $y1, $x2, $y2, $x3, $y3)` (alias `circ3`) — finds the unique circle passing through three non-collinear points. Returns [$cx, $cy, $radius] or undef if collinear.

```
my $c = circ3(0, 0, 1, 0, 0.5, 0.866)
p "center=($c->[0], $c->[1]) r=$c->[2]"
```

circle_of_fifths_step

`circle_of_fifths_step` — batch 9 builtin.

```
my $result = circle_of_fifths_step $x
# or in a pipeline:
@list |> map circle_of_fifths_step |> p
```

circuit_depth

`circuit_depth` — batch 80 builtin.

```
my $result = circuit_depth $x
# or in a pipeline:
@list |> map circuit_depth |> p
```

circuit_width

`circuit_width` — batch 80 builtin.

```
my $result = circuit_width $x
# or in a pipeline:
@list |> map circuit_width |> p
```

circum

`circum` — geometry / physics builtin. Alias for `circumference`.

```
my $result = circum $x
# or in a pipeline:
@list |> map circum |> p
```

circumference

`circumference` — geometry / physics builtin.

```
my $result = circumference $x
# or in a pipeline:
@list |> map circumference |> p
```

cityhash64

`cityhash64` — batch 81 builtin.

```
my $result = cityhash64 $x
# or in a pipeline:
@list |> map cityhash64 |> p
```

clamp_array

`clamp_array` — matrix operations (uncategorized batch) builtin.

```
my $result = clamp_array $x
# or in a pipeline:
@list |> map clamp_array |> p
```

clamp_each

`clamp_each` — conversion / utility (batch 4) builtin.

```
my $result = clamp_each $x
# or in a pipeline:
@list |> map clamp_each |> p
```

clamp_fn

`clamp_fn` — functional combinators builtin.

```
my $result = clamp_fn $x
# or in a pipeline:
@list |> map clamp_fn |> p
```

clamp_int

`clamp_int` — extended stdlib builtin.

```
my $result = clamp_int $x
# or in a pipeline:
@list |> map clamp_int |> p
```

clamp_list

`clamp_list` — functional combinators builtin.

```
my $result = clamp_list $x
# or in a pipeline:
@list |> map clamp_list |> p
```

clarr

`clarr` — matrix operations (uncategorized batch) builtin. Alias for `clamp_array`.

```
my $result = clarr $x
# or in a pipeline:
@list |> map clarr |> p
```

class

`class` declares a full object-oriented class with typed fields, inheritance, traits, instance methods, and static methods. Classes provide modern OOP semantics with a clean syntax.

Declaration:

```
class Animal {
    name: Str
    age: Int = 0
    fn speak { p "Animal: " . $self->name }
}
```

Inheritance with `extends`:

```
class Dog extends Animal {
    breed: Str = "Mixed"
    fn bark { p "Woof! I am " . $self->name }
    fn speak { p $self->name . " barks!" }  # override
}
```

Construction (named or positional):

```
my $dog = Dog(name => "Rex", age => 5, breed => "Lab")
my $dog = Dog("Rex", 5, "Lab")  # positional
```

Field access (getter/setter):

```
p $dog->name      # getter
$dog->age(6)      # setter
```

Static methods (`fn Self.name`):

```
class Math {
    fn Self.add($a, $b) { $a + $b }
    fn Self.pi { 3.14159 }
}
p Math::add(3, 4)  # 7
p Math::pi()       # 3.14159
```

Traits (interfaces):

```
trait Printable { fn to_str }
class Item impl Printable {
    name: Str
    fn to_str { $self->name }
}
```

Multiple inheritance:

```
class C extends A, B { }
```

isa checks:

```
p $dog->isa("Dog")    # 1
p $dog->isa("Animal") # 1 (parent)
p $dog->isa("Cat")    # "" (false)
```

Built-in methods:

```
my @f = $dog->fields()        # (name, age, breed)
my $h = $dog->to_hash()       # { name => "Rex", ... }
my $d2 = $dog->with(age => 1) # functional update
my $d3 = $dog->clone()        # deep copy
```

Visibility (pub/priv):

```
class Secret {
    pub visible: Int = 1
    priv hidden: Int = 42
    pub fn get_hidden { $self->hidden }  # internal access ok
}
```

Note: inherited fields come first in the values array; method lookup walks the inheritance chain.

class_number_bound

`class_number_bound` — batch 35 builtin.

```
my $result = class_number_bound $x
# or in a pipeline:
@list |> map class_number_bound |> p
```

clausen_cl2

`clausen_cl2` — batch 29 builtin.

```
my $result = clausen_cl2 $x
# or in a pipeline:
@list |> map clausen_cl2 |> p
```

clausius_clapeyron

`clausius_clapeyron` — batch 21 builtin.

```
my $result = clausius_clapeyron $x
# or in a pipeline:
@list |> map clausius_clapeyron |> p
```

clausius_clapeyron_full

`clausius_clapeyron_full` — batch 42 builtin.

```
my $result = clausius_clapeyron_full $x
# or in a pipeline:
@list |> map clausius_clapeyron_full |> p
```

clean_price

`clean_price` — batch 57 builtin.

```
my $result = clean_price $x
# or in a pipeline:
@list |> map clean_price |> p
```

clighten

`clighten` — color operations builtin. Alias for `color_lighten`.

```
my $result = clighten $x
# or in a pipeline:
@list |> map clighten |> p
```

clique_count_3

`clique_count_3` — batches 14-16 builtin.

```
my $result = clique_count_3 $x
# or in a pipeline:
@list |> map clique_count_3 |> p
```

clique_number_lower

`clique_number_lower` — batch 41 builtin.

```
my $result = clique_number_lower $x
# or in a pipeline:
@list |> map clique_number_lower |> p
```

closed_pipe_harmonic

`closed_pipe_harmonic` — batch 23 builtin.

```
my $result = closed_pipe_harmonic $x
# or in a pipeline:
@list |> map closed_pipe_harmonic |> p
```

closest_pt_segment_2d

`closest_pt_segment_2d` — batch 28 builtin.

```
my $result = closest_pt_segment_2d $x
# or in a pipeline:
@list |> map closest_pt_segment_2d |> p
```

clpi

`clpi` — extended stdlib builtin. Alias for `clamp_int`.

```
my $result = clpi $x
# or in a pipeline:
@list |> map clpi |> p
```

cluster_attack_rate

`cluster_attack_rate` — batch 61 builtin.

```
my $result = cluster_attack_rate $x
# or in a pipeline:
@list |> map cluster_attack_rate |> p
```

cluster_slots

`cluster_slots` — batch 71 builtin.

```
my $result = cluster_slots $x
# or in a pipeline:
@list |> map cluster_slots |> p
```

clustering_coefficient

`clustering_coefficient` — batch 75 builtin.

```
my $result = clustering_coefficient $x
# or in a pipeline:
@list |> map clustering_coefficient |> p
```

clutch_score

`clutch_score` — batch 56 builtin.

```
my $result = clutch_score $x
# or in a pipeline:
@list |> map clutch_score |> p
```

cm_to_inches

`cm_to_inches` — unit conversions builtin.

```
my $result = cm_to_inches $input
```

cmap

`cmap` — additional missing stdlib functions builtin. Alias for `concat_map`.

```
my $result = cmap $x
# or in a pipeline:
@list |> map cmap |> p
```

cmb_temperature

`cmb_temperature` — batch 31 builtin.

```
my $result = cmb_temperature $x
# or in a pipeline:
@list |> map cmb_temperature |> p
```

cmb_temperature_at_z

`cmb_temperature_at_z` — batch 31 builtin.

```
my $result = cmb_temperature_at_z $x
# or in a pipeline:
@list |> map cmb_temperature_at_z |> p
```

cmd_exists

`cmd_exists` — process / env builtin.

```
my $result = cmd_exists $x
# or in a pipeline:
@list |> map cmd_exists |> p
```

cmp_num

`cmp_num` — file stat / path builtin. Alias for `spaceship`.

```
my $result = cmp_num $x
# or in a pipeline:
@list |> map cmp_num |> p
```

cmp_str

`cmp_str` — file stat / path builtin.

```
my $result = cmp_str $x
# or in a pipeline:
@list |> map cmp_str |> p
```

cmpr

`cmpr` — python/ruby stdlib builtin. Alias for `compress`.

```
my $result = cmpr $x
# or in a pipeline:
@list |> map cmpr |> p
```

cmyk_to_rgb

`cmyk_to_rgb` — batch 82 builtin.

```
my $result = cmyk_to_rgb $input
```

cn_coefficient

`cn_coefficient` — batch 34 builtin.

```
my $result = cn_coefficient $x
# or in a pipeline:
@list |> map cn_coefficient |> p
```

cn_ode

`cn_ode` — batch 19 builtin. Alias for `crank_nicolson_ode`.

```
my $result = cn_ode $x
# or in a pipeline:
@list |> map cn_ode |> p
```

cnf_dpll_branch

`cnf_dpll_branch` — batch 67 builtin.

```
my $result = cnf_dpll_branch $x
# or in a pipeline:
@list |> map cnf_dpll_branch |> p
```

cnf_pure_literal_elim

`cnf_pure_literal_elim` — batch 67 builtin.

```
my $result = cnf_pure_literal_elim $x
# or in a pipeline:
@list |> map cnf_pure_literal_elim |> p
```

cnf_unit_propagate

`cnf_unit_propagate` — batch 67 builtin.

```
my $result = cnf_unit_propagate $x
# or in a pipeline:
@list |> map cnf_unit_propagate |> p
```

cnt

`count` (aliases `len`, `cnt`) returns the number of elements in a list, the codepoint count of a string, the number of key-value pairs in a hash, or the cardinality of a set. Universal length-like function that dispatches on the type of its argument. Replaces `scalar @array` / `length $string` for stryke code; on strings it pairs with `[i]` / `[a:b]` and `cindex` / `crindex`, all of which use codepoint positions.

For byte length on a string, use Perl 5's `length` (kept byte-based for compat with `index` / `rindex` / `substr`). `size` is NOT a count alias — it returns a file's byte size.

```
my @arr = (1, 2, 3, 4, 5)
p cnt @arr          # 5
p len "hello"       # 5
p len "🦓"          # 1 (codepoint)
p length "🦓"       # 3 (bytes)
my %h = (a => 1, b => 2)
p cnt \%h           # 2
rl("file.txt") |> cnt |> p  # line count
```

cntw

`cntw` — ruby enumerable extras builtin. Alias for `count_while`.

```
my $result = cntw $x
# or in a pipeline:
@list |> map cntw |> p
```

co2_growth_rate_step

`co2_growth_rate_step` — batch 42 builtin.

```
my $result = co2_growth_rate_step $x
# or in a pipeline:
@list |> map co2_growth_rate_step |> p
```

coalesce

`coalesce` — functional combinators builtin.

```
my $result = coalesce $x
# or in a pipeline:
@list |> map coalesce |> p
```

coalescent_expected_time

`coalescent_expected_time` — batch 33 builtin.

```
my $result = coalescent_expected_time $x
# or in a pipeline:
@list |> map coalescent_expected_time |> p
```

coalescent_tree_length

`coalescent_tree_length` — batch 33 builtin.

```
my $result = coalescent_tree_length $x
# or in a pipeline:
@list |> map coalescent_tree_length |> p
```

cobb_douglas

`cobb_douglas` — batch 73 builtin.

```
my $result = cobb_douglas $x
# or in a pipeline:
@list |> map cobb_douglas |> p
```

code_point_at

`code_point_at` — extended stdlib builtin.

```
my $result = code_point_at $x
# or in a pipeline:
@list |> map code_point_at |> p
```

codifferential_step

`codifferential_step` — batch 39 builtin.

```
my $result = codifferential_step $x
# or in a pipeline:
@list |> map codifferential_step |> p
```

codon_adaptation_index

`codon_adaptation_index` — batch 33 builtin.

```
my $result = codon_adaptation_index $x
# or in a pipeline:
@list |> map codon_adaptation_index |> p
```

codon_usage_variance

`codon_usage_variance` — batch 33 builtin.

```
my $result = codon_usage_variance $x
# or in a pipeline:
@list |> map codon_usage_variance |> p
```

coeff_of_variation

`coeff_of_variation` — math / numeric (uncategorized batch) builtin.

```
my $result = coeff_of_variation $x
# or in a pipeline:
@list |> map coeff_of_variation |> p
```

cognitive_hierarchy_step

`cognitive_hierarchy_step` — batch 43 builtin.

```
my $result = cognitive_hierarchy_step $x
# or in a pipeline:
@list |> map cognitive_hierarchy_step |> p
```

cohen_d

`cohen_d` (alias `cohend`) computes Cohen's d effect size between two samples. Small=0.2, medium=0.5, large=0.8.

```
p cohend([1,2,3], [4,5,6])  # large effect
```

cohen_kappa

`cohen_kappa` — batch 27 builtin.

```
my $result = cohen_kappa $x
# or in a pipeline:
@list |> map cohen_kappa |> p
```

coherence_xy

`coherence_xy` — batch 74 builtin.

```
my $result = coherence_xy $x
# or in a pipeline:
@list |> map coherence_xy |> p
```

coherent_mean_photons

`coherent_mean_photons` — batch 32 builtin.

```
my $result = coherent_mean_photons $x
# or in a pipeline:
@list |> map coherent_mean_photons |> p
```

cohomology_rank

`cohomology_rank` — batch 37 builtin.

```
my $result = cohomology_rank $x
# or in a pipeline:
@list |> map cohomology_rank |> p
```

coif1_coeffs

`coif1_coeffs` — batches 14-16 builtin.

```
my $result = coif1_coeffs $x
# or in a pipeline:
@list |> map coif1_coeffs |> p
```

coiflet_wavelet_step

`coiflet_wavelet_step` — batch 40 builtin.

```
my $result = coiflet_wavelet_step $x
# or in a pipeline:
@list |> map coiflet_wavelet_step |> p
```

coin_flip

`coin_flip` — random builtin.

```
my $result = coin_flip $x
# or in a pipeline:
@list |> map coin_flip |> p
```

cointegration_eg

`cointegration_eg` — batch 78 builtin.

```
my $result = cointegration_eg $x
# or in a pipeline:
@list |> map cointegration_eg |> p
```

cointegration_residual

`cointegration_residual` — batch 36 builtin.

```
my $result = cointegration_residual $x
# or in a pipeline:
@list |> map cointegration_residual |> p
```

colMeans

`col_means` (alias `colMeans`) — mean of each column. Like R's colMeans().

```
p colMeans([[2,4],[6,8]])  # [4, 6]
```

colSums

`col_sums` (alias `colSums`) — sum of each column. Like R's colSums().

```
p colSums([[1,2],[3,4]])  # [4, 6]
```

cole_cole_eis

`cole_cole_eis` — batch 38 builtin.

```
my $result = cole_cole_eis $x
# or in a pipeline:
@list |> map cole_cole_eis |> p
```

collapse_whitespace

`collapse_whitespace` — string processing (uncategorized batch) builtin.

```
my $result = collapse_whitespace $x
# or in a pipeline:
@list |> map collapse_whitespace |> p
```

collatz

`collatz` — math / numeric (uncategorized batch) builtin. Alias for `collatz_length`.

```
my $result = collatz $x
# or in a pipeline:
@list |> map collatz |> p
```

collatz_length

`collatz_length` — math / numeric (uncategorized batch) builtin.

```
my $result = collatz_length $x
# or in a pipeline:
@list |> map collatz_length |> p
```

collatz_sequence

`collatz_sequence` — math / numeric (uncategorized batch) builtin.

```
my $result = collatz_sequence $x
# or in a pipeline:
@list |> map collatz_sequence |> p
```

collatzseq

`collatzseq` — math / numeric (uncategorized batch) builtin. Alias for `collatz_sequence`.

```
my $result = collatzseq $x
# or in a pipeline:
@list |> map collatzseq |> p
```

collusion_payoff_step

`collusion_payoff_step` — batch 43 builtin.

```
my $result = collusion_payoff_step $x
# or in a pipeline:
@list |> map collusion_payoff_step |> p
```

collws

`collws` — string processing (uncategorized batch) builtin. Alias for `collapse_whitespace`.

```
my $result = collws $x
# or in a pipeline:
@list |> map collws |> p
```

color256

`color256` — color / ansi builtin.

```
my $result = color256 $x
# or in a pipeline:
@list |> map color256 |> p
```

color_blend

`color_blend` — color operations builtin.

```
my $result = color_blend $x
# or in a pipeline:
@list |> map color_blend |> p
```

color_blend_alpha

`color_blend_alpha` — batch 82 builtin.

```
my $result = color_blend_alpha $x
# or in a pipeline:
@list |> map color_blend_alpha |> p
```

color_blend_t

`color_blend_t` — batch 9 builtin.

```
my $result = color_blend_t $x
# or in a pipeline:
@list |> map color_blend_t |> p
```

color_complement

`color_complement` — color operations builtin.

```
my $result = color_complement $x
# or in a pipeline:
@list |> map color_complement |> p
```

color_darken

`color_darken` — color operations builtin.

```
my $result = color_darken $x
# or in a pipeline:
@list |> map color_darken |> p
```

color_distance

`color_distance` — color operations builtin.

```
my $result = color_distance $x
# or in a pipeline:
@list |> map color_distance |> p
```

color_grayscale

`color_grayscale` — color operations builtin.

```
my $result = color_grayscale $x
# or in a pipeline:
@list |> map color_grayscale |> p
```

color_invert

`color_invert` — color operations builtin.

```
my $result = color_invert $x
# or in a pipeline:
@list |> map color_invert |> p
```

color_lighten

`color_lighten` — color operations builtin.

```
my $result = color_lighten $x
# or in a pipeline:
@list |> map color_lighten |> p
```

color_temperature_kelvin

`color_temperature_kelvin` — batch 82 builtin.

```
my $result = color_temperature_kelvin $x
# or in a pipeline:
@list |> map color_temperature_kelvin |> p
```

columbic_capacity_lihalfcell

`columbic_capacity_lihalfcell` — batch 38 builtin.

```
my $result = columbic_capacity_lihalfcell $x
# or in a pipeline:
@list |> map columbic_capacity_lihalfcell |> p
```

column_generation_step

`column_generation_step` — batch 41 builtin.

```
my $result = column_generation_step $x
# or in a pipeline:
@list |> map column_generation_step |> p
```

comb

`combinations($n, $r)` (alias `comb`) — number of ways to choose r items from n. Formula: n!/(r!(n-r)!). Order doesn't matter.

```
p comb(5, 3)    # 10 (ways to choose 3 items from 5)
p comb(52, 5)   # 2598960 (poker hands)
```

combinations_rep

`combinations_rep` — additional missing stdlib functions builtin.

```
my $result = combinations_rep $x
# or in a pipeline:
@list |> map combinations_rep |> p
```

combinatorial_auction_step

`combinatorial_auction_step` — batch 43 builtin.

```
my $result = combinatorial_auction_step $x
# or in a pipeline:
@list |> map combinatorial_auction_step |> p
```

combrep

`combrep` — additional missing stdlib functions builtin. Alias for `combinations_rep`.

```
my $result = combrep $x
# or in a pipeline:
@list |> map combrep |> p
```

common_kmers

`common_kmers` — batch 33 builtin.

```
my $result = common_kmers $x
# or in a pipeline:
@list |> map common_kmers |> p
```

common_knowledge_iterations

`common_knowledge_iterations` — batch 43 builtin.

```
my $result = common_knowledge_iterations $x
# or in a pipeline:
@list |> map common_knowledge_iterations |> p
```

common_prefix

`common_prefix` — extended stdlib builtin.

```
my $result = common_prefix $x
# or in a pipeline:
@list |> map common_prefix |> p
```

common_suffix

`common_suffix` — extended stdlib builtin.

```
my $result = common_suffix $x
# or in a pipeline:
@list |> map common_suffix |> p
```

comoving_distance

`comoving_distance` — batch 31 builtin.

```
my $result = comoving_distance $x
# or in a pipeline:
@list |> map comoving_distance |> p
```

comoving_distance_approx

`comoving_distance_approx` — batch 30 builtin.

```
my $result = comoving_distance_approx $x
# or in a pipeline:
@list |> map comoving_distance_approx |> p
```

compare_versions

`compare_versions` — file stat / path builtin.

```
my $result = compare_versions $x
# or in a pipeline:
@list |> map compare_versions |> p
```

composition_count

`composition_count` — batch 9 builtin.

```
my $result = composition_count $x
# or in a pipeline:
@list |> map composition_count |> p
```

compound_interest

`compound_interest` — physics formulas builtin.

```
my $result = compound_interest $x
# or in a pipeline:
@list |> map compound_interest |> p
```

compound_interest_periods

`compound_interest_periods` — batch 13 builtin.

```
my $result = compound_interest_periods $x
# or in a pipeline:
@list |> map compound_interest_periods |> p
```

compress

`compress` — python/ruby stdlib builtin.

```
my $result = compress $x
# or in a pipeline:
@list |> map compress |> p
```

compressibility_z

`compressibility_z` — batch 21 builtin.

```
my $result = compressibility_z $x
# or in a pipeline:
@list |> map compressibility_z |> p
```

compton_wavelength

`compton_wavelength` — batch 32 builtin.

```
my $result = compton_wavelength $x
# or in a pipeline:
@list |> map compton_wavelength |> p
```

compute_residuals

`compute_residuals` — batch 9 builtin. Alias for `residuals_compute`.

```
my $result = compute_residuals $x
# or in a pipeline:
@list |> map compute_residuals |> p
```

concat_map

`concat_map` — additional missing stdlib functions builtin.

```
my $result = concat_map $x
# or in a pipeline:
@list |> map concat_map |> p
```

concentration_dilute

`concentration_dilute` — batch 12 builtin.

```
my $result = concentration_dilute $x
# or in a pipeline:
@list |> map concentration_dilute |> p
```

concordance_correlation

`concordance_correlation` — batches 14-16 builtin.

```
my $result = concordance_correlation $x
# or in a pipeline:
@list |> map concordance_correlation |> p
```

cond

`matrix_cond` (aliases `mcond`, `cond`) estimates the condition number of a matrix (ratio of largest to smallest singular value). Large values indicate ill-conditioning.

```
p cond([[1,0],[0,1]])    # 1 (perfect)
p cond([[1,2],[2,4]])    # Inf (singular)
```

conditional_entropy_step

`conditional_entropy_step` — batch 40 builtin.

```
my $result = conditional_entropy_step $x
# or in a pipeline:
@list |> map conditional_entropy_step |> p
```

conditional_gradient_step

`conditional_gradient_step` — batch 41 builtin.

```
my $result = conditional_gradient_step $x
# or in a pipeline:
@list |> map conditional_gradient_step |> p
```

condorcet_winner_check

`condorcet_winner_check` — batch 43 builtin.

```
my $result = condorcet_winner_check $x
# or in a pipeline:
@list |> map condorcet_winner_check |> p
```

cone_surface

`cone_surface` — batch 28 builtin.

```
my $result = cone_surface $x
# or in a pipeline:
@list |> map cone_surface |> p
```

cone_volume

`cone_volume` — geometry / physics builtin.

```
my $result = cone_volume $x
# or in a pipeline:
@list |> map cone_volume |> p
```

conevol

`conevol` — geometry / physics builtin. Alias for `cone_volume`.

```
my $result = conevol $x
# or in a pipeline:
@list |> map conevol |> p
```

confint

`confint_lm` (alias `confint`) — confidence intervals for model coefficients. Returns hash with intercept_lower/upper, slope_lower/upper.

```
my $ci = confint(lm([1,2,3,4,5], [2,4,5,4,5]))
p $ci->{slope_lower}
```

conformal_compactification_step

`conformal_compactification_step` — batch 39 builtin.

```
my $result = conformal_compactification_step $x
# or in a pipeline:
@list |> map conformal_compactification_step |> p
```

confusion_counts

`confusion_counts` — batch 27 builtin.

```
my $result = confusion_counts $x
# or in a pipeline:
@list |> map confusion_counts |> p
```

conj

`conj` — algebraic match builtin.

```
my $result = conj $x
# or in a pipeline:
@list |> map conj |> p
```

connected_components

`connected_components` — batch 13 builtin.

```
my $result = connected_components $x
# or in a pipeline:
@list |> map connected_components |> p
```

connected_components_graph

`connected_components_graph` — extended stdlib builtin.

```
my $result = connected_components_graph $x
# or in a pipeline:
@list |> map connected_components_graph |> p
```

cons

`cons` — algebraic match builtin.

```
my $result = cons $x
# or in a pipeline:
@list |> map cons |> p
```

consecutive_eq

`consecutive_eq` — collection more builtin.

```
my $result = consecutive_eq $x
# or in a pipeline:
@list |> map consecutive_eq |> p
```

consecutive_pairs

`consecutive_pairs` — list helpers (batch 4) builtin.

```
my $result = consecutive_pairs $x
# or in a pipeline:
@list |> map consecutive_pairs |> p
```

const_fn

`const_fn` — functional primitives builtin.

```
my $result = const_fn $x
# or in a pipeline:
@list |> map const_fn |> p
```

constant_case

`constant_case` — string (batch 2) builtin.

```
my $result = constant_case $x
# or in a pipeline:
@list |> map constant_case |> p
```

constituency_cyk

`constituency_cyk` — batch 69 builtin.

```
my $result = constituency_cyk $x
# or in a pipeline:
@list |> map constituency_cyk |> p
```

consumer_surplus

`consumer_surplus` — batch 73 builtin.

```
my $result = consumer_surplus $x
# or in a pipeline:
@list |> map consumer_surplus |> p
```

contact_tracing_eff

`contact_tracing_eff` — batch 61 builtin.

```
my $result = contact_tracing_eff $x
# or in a pipeline:
@list |> map contact_tracing_eff |> p
```

contains

`contains` — trivial string ops builtin.

```
my $result = contains $x
# or in a pipeline:
@list |> map contains |> p
```

contains_all

`contains_all` — string (batch 2) builtin.

```
my $result = contains_all $x
# or in a pipeline:
@list |> map contains_all |> p
```

contains_any

`contains_any` — string (batch 2) builtin.

```
my $result = contains_any $x
# or in a pipeline:
@list |> map contains_any |> p
```

contains_elem

`contains_elem` — list helpers (batch 4) builtin.

```
my $result = contains_elem $x
# or in a pipeline:
@list |> map contains_elem |> p
```

contcomp

`contcomp` — finance (extended) builtin. Alias for `continuous_compound`.

```
my $result = contcomp $x
# or in a pipeline:
@list |> map contcomp |> p
```

content_negotiate

`content_negotiate` — batch 81 builtin.

```
my $result = content_negotiate $x
# or in a pipeline:
@list |> map content_negotiate |> p
```

continued_fraction_sqrt

`continued_fraction_sqrt` — batch 35 builtin.

```
my $result = continued_fraction_sqrt $x
# or in a pipeline:
@list |> map continued_fraction_sqrt |> p
```

continuous_compound

`continuous_compound($principal, $rate, $time)` (alias `ccomp`) — computes future value with continuous compounding: P × e^(rt).

```
p ccomp(1000, 0.05, 10)   # ~1648.72
```

contrast_ratio

`contrast_ratio` — batch 82 builtin.

```
my $result = contrast_ratio $x
# or in a pipeline:
@list |> map contrast_ratio |> p
```

contrast_ratio_wcag

`contrast_ratio_wcag` — batch 9 builtin.

```
my $result = contrast_ratio_wcag $x
# or in a pipeline:
@list |> map contrast_ratio_wcag |> p
```

conv

`conv` — math / numeric (uncategorized batch) builtin. Alias for `convolution`.

```
my $result = conv $x
# or in a pipeline:
@list |> map conv |> p
```

converge

`converge` — functional combinators builtin.

```
my $result = converge $x
# or in a pipeline:
@list |> map converge |> p
```

convex_hull_chan

`convex_hull_chan` — batch 58 builtin.

```
my $result = convex_hull_chan $x
# or in a pipeline:
@list |> map convex_hull_chan |> p
```

convex_hull_jarvis

`convex_hull_jarvis` — batch 28 builtin.

```
my $result = convex_hull_jarvis $x
# or in a pipeline:
@list |> map convex_hull_jarvis |> p
```

convexity

`convexity` — batch 57 builtin.

```
my $result = convexity $x
# or in a pipeline:
@list |> map convexity |> p
```

convolution

`convolution` — math / numeric (uncategorized batch) builtin.

```
my $result = convolution $x
# or in a pipeline:
@list |> map convolution |> p
```

convolve_2d

`convolve_2d` — batch 77 builtin.

```
my $result = convolve_2d $x
# or in a pipeline:
@list |> map convolve_2d |> p
```

convolve_full

`convolve_full` — batch 72 builtin.

```
my $result = convolve_full $x
# or in a pipeline:
@list |> map convolve_full |> p
```

convolve_valid

`convolve_valid` — batch 72 builtin.

```
my $result = convolve_valid $x
# or in a pipeline:
@list |> map convolve_valid |> p
```

coombs_runoff_step

`coombs_runoff_step` — batch 43 builtin.

```
my $result = coombs_runoff_step $x
# or in a pipeline:
@list |> map coombs_runoff_step |> p
```

cooperative_game_value

`cooperative_game_value` — batch 43 builtin.

```
my $result = cooperative_game_value $x
# or in a pipeline:
@list |> map cooperative_game_value |> p
```

coordination_game_payoff

`coordination_game_payoff` — batch 43 builtin.

```
my $result = coordination_game_payoff $x
# or in a pipeline:
@list |> map coordination_game_payoff |> p
```

copeland_score_step

`copeland_score_step` — batch 43 builtin.

```
my $result = copeland_score_step $x
# or in a pipeline:
@list |> map copeland_score_step |> p
```

coppersmith_bound

`coppersmith_bound` — batch 35 builtin.

```
my $result = coppersmith_bound $x
# or in a pipeline:
@list |> map coppersmith_bound |> p
```

coptic_from_fixed

`coptic_from_fixed` — batch 52 builtin.

```
my $result = coptic_from_fixed $x
# or in a pipeline:
@list |> map coptic_from_fixed |> p
```

copy_sign

`copy_sign` — extended stdlib builtin.

```
my $result = copy_sign $x
# or in a pipeline:
@list |> map copy_sign |> p
```

copy_within

`copy_within` — additional missing stdlib functions builtin.

```
my $result = copy_within $x
# or in a pipeline:
@list |> map copy_within |> p
```

copysign

`copysign` — math functions builtin.

```
my $result = copysign $x
# or in a pipeline:
@list |> map copysign |> p
```

coq_tactic_apply

`coq_tactic_apply` — batch 67 builtin.

```
my $result = coq_tactic_apply $x
# or in a pipeline:
@list |> map coq_tactic_apply |> p
```

coq_unify_term

`coq_unify_term` — batch 67 builtin.

```
my $result = coq_unify_term $x
# or in a pipeline:
@list |> map coq_unify_term |> p
```

cor_mat

`cor_matrix` (alias `cor_mat`) — correlation matrix from observations. Each row is an observation vector.

```
my $R = cor_mat([[1,2],[3,4],[5,6]])  # 2x2 correlation matrix
```

core_coalition

`core_coalition` — batch 73 builtin.

```
my $result = core_coalition $x
# or in a pipeline:
@list |> map core_coalition |> p
```

core_membership_check

`core_membership_check` — batch 43 builtin.

```
my $result = core_membership_check $x
# or in a pipeline:
@list |> map core_membership_check |> p
```

coreference_singleton

`coreference_singleton` — batch 69 builtin.

```
my $result = coreference_singleton $x
# or in a pipeline:
@list |> map coreference_singleton |> p
```

corner_subpix

`corner_subpix` — batch 77 builtin.

```
my $result = corner_subpix $x
# or in a pipeline:
@list |> map corner_subpix |> p
```

corr

`corr` — extended stdlib builtin. Alias for `correlation`.

```
my $result = corr $x
# or in a pipeline:
@list |> map corr |> p
```

corr_agg

`corr_agg` — batch 70 builtin.

```
my $result = corr_agg $x
# or in a pipeline:
@list |> map corr_agg |> p
```

correlate_full

`correlate_full` — batch 72 builtin.

```
my $result = correlate_full $x
# or in a pipeline:
@list |> map correlate_full |> p
```

correlated_equilibrium_value

`correlated_equilibrium_value` — batch 43 builtin.

```
my $result = correlated_equilibrium_value $x
# or in a pipeline:
@list |> map correlated_equilibrium_value |> p
```

correlation

`correlation` — extended stdlib builtin.

```
my $result = correlation $x
# or in a pipeline:
@list |> map correlation |> p
```

corsi_for

`corsi_for` — batch 56 builtin.

```
my $result = corsi_for $x
# or in a pipeline:
@list |> map corsi_for |> p
```

cosh

`cosh` — trig / math (batch 2) builtin.

```
my $result = cosh $x
# or in a pipeline:
@list |> map cosh |> p
```

cosh_minus1_over_x2

`cosh_minus1_over_x2` — batch 29 builtin.

```
my $result = cosh_minus1_over_x2 $x
# or in a pipeline:
@list |> map cosh_minus1_over_x2 |> p
```

cosine_integral_ci

`cosine_integral_ci` — batch 29 builtin.

```
my $result = cosine_integral_ci $x
# or in a pipeline:
@list |> map cosine_integral_ci |> p
```

cosine_similarity

`cosine_similarity` — math functions builtin.

```
my $result = cosine_similarity $x
# or in a pipeline:
@list |> map cosine_similarity |> p
```

cosmic_distance_modulus_si

`cosmic_distance_modulus_si` — batch 31 builtin.

```
my $result = cosmic_distance_modulus_si $x
# or in a pipeline:
@list |> map cosmic_distance_modulus_si |> p
```

cosmological_constant

`cosmological_constant` — batch 31 builtin.

```
my $result = cosmological_constant $x
# or in a pipeline:
@list |> map cosmological_constant |> p
```

cosmological_constant_term

`cosmological_constant_term` — batch 39 builtin.

```
my $result = cosmological_constant_term $x
# or in a pipeline:
@list |> map cosmological_constant_term |> p
```

cot

`cot` returns the cotangent (1/tan) of an angle in radians.

```
p cot(0.7854)  # ≈ 1.0 (cot 45°)
```

coulomb

`coulomb_force($q1, $q2, $r)` (alias `coulomb`) — Coulomb's law: $F = kq_1q_2/r^2$. Charges in Coulombs, distance in meters.

```
p coulomb(1e-6, 1e-6, 0.1)   # ~0.9 N (two 1μC charges 10cm apart)
```

coulomb_constant

`coulomb_constant` — physics constants builtin.

```
my $result = coulomb_constant $x
# or in a pipeline:
@list |> map coulomb_constant |> p
```

coulomb_f0

`coulomb_f0` — batch 29 builtin.

```
my $result = coulomb_f0 $x
# or in a pipeline:
@list |> map coulomb_f0 |> p
```

coulombic_efficiency_cell

`coulombic_efficiency_cell` — batch 38 builtin.

```
my $result = coulombic_efficiency_cell $x
# or in a pipeline:
@list |> map coulombic_efficiency_cell |> p
```

count_by

`count_by` — functional / iterator builtin. Alias for `take_while`.

```
my $result = count_by $x
# or in a pipeline:
@list |> map count_by |> p
```

count_bytes

`count_bytes` — extended stdlib batch 3 builtin.

```
my $result = count_bytes $x
# or in a pipeline:
@list |> map count_bytes |> p
```

count_char

`count_char` — string (batch 2) builtin.

```
my $result = count_char $x
# or in a pipeline:
@list |> map count_char |> p
```

count_chars

`count_chars` — extended stdlib batch 3 builtin.

```
my $result = count_chars $x
# or in a pipeline:
@list |> map count_chars |> p
```

count_consonants

`count_consonants` — string (batch 2) builtin.

```
my $result = count_consonants $x
# or in a pipeline:
@list |> map count_consonants |> p
```

count_digits

`count_digits` — counters (batch 4) builtin.

```
my $result = count_digits $x
# or in a pipeline:
@list |> map count_digits |> p
```

count_elem

`count_elem` — list helpers (batch 4) builtin.

```
my $result = count_elem $x
# or in a pipeline:
@list |> map count_elem |> p
```

count_eq

`count_eq` — more list helpers builtin.

```
my $result = count_eq $x
# or in a pipeline:
@list |> map count_eq |> p
```

count_inversions

`count_inversions` — extended stdlib batch 3 builtin.

```
my $result = count_inversions $x
# or in a pipeline:
@list |> map count_inversions |> p
```

count_letters

`count_letters` — counters (batch 4) builtin.

```
my $result = count_letters $x
# or in a pipeline:
@list |> map count_letters |> p
```

count_lines

`count_lines` — extended stdlib batch 3 builtin.

```
my $result = count_lines $x
# or in a pipeline:
@list |> map count_lines |> p
```

count_lower

`count_lower` — counters (batch 4) builtin.

```
my $result = count_lower $x
# or in a pipeline:
@list |> map count_lower |> p
```

count_match

`count_match` — counters (batch 4) builtin.

```
my $result = count_match $x
# or in a pipeline:
@list |> map count_match |> p
```

count_ne

`count_ne` — more list helpers builtin.

```
my $result = count_ne $x
# or in a pipeline:
@list |> map count_ne |> p
```

count_punctuation

`count_punctuation` — counters (batch 4) builtin.

```
my $result = count_punctuation $x
# or in a pipeline:
@list |> map count_punctuation |> p
```

count_regex_matches

`count_regex_matches` — file stat / path builtin.

```
my $result = count_regex_matches $x
# or in a pipeline:
@list |> map count_regex_matches |> p
```

count_spaces

`count_spaces` — counters (batch 4) builtin.

```
my $result = count_spaces $x
# or in a pipeline:
@list |> map count_spaces |> p
```

count_substring

`count_substring` — file stat / path builtin.

```
my $result = count_substring $x
# or in a pipeline:
@list |> map count_substring |> p
```

count_upper

`count_upper` — counters (batch 4) builtin.

```
my $result = count_upper $x
# or in a pipeline:
@list |> map count_upper |> p
```

count_vowels

`count_vowels` — string (batch 2) builtin.

```
my $result = count_vowels $x
# or in a pipeline:
@list |> map count_vowels |> p
```

count_while

`count_while` — ruby enumerable extras builtin.

```
my $result = count_while $x
# or in a pipeline:
@list |> map count_while |> p
```

count_words

`count_words` — extended stdlib batch 3 builtin.

```
my $result = count_words $x
# or in a pipeline:
@list |> map count_words |> p
```

counter

`counter` — data structure helpers builtin.

```
my $result = counter $x
# or in a pipeline:
@list |> map counter |> p
```

counter_most_common

`counter_most_common` — data structure helpers builtin.

```
my $result = counter_most_common $x
# or in a pipeline:
@list |> map counter_most_common |> p
```

countif

`countif` — batch 57 builtin.

```
my $result = countif $x
# or in a pipeline:
@list |> map countif |> p
```

countifs

`countifs` — batch 57 builtin.

```
my $result = countifs $x
# or in a pipeline:
@list |> map countifs |> p
```

coupon_count

`coupon_count` — batch 57 builtin.

```
my $result = coupon_count $x
# or in a pipeline:
@list |> map coupon_count |> p
```

courant_friedrichs_lewy

`courant_friedrichs_lewy` — batch 42 builtin.

```
my $result = courant_friedrichs_lewy $x
# or in a pipeline:
@list |> map courant_friedrichs_lewy |> p
```

cournot_eq

`cournot_eq` — batch 73 builtin.

```
my $result = cournot_eq $x
# or in a pipeline:
@list |> map cournot_eq |> p
```

cournot_quantity_step

`cournot_quantity_step` — batch 43 builtin.

```
my $result = cournot_quantity_step $x
# or in a pipeline:
@list |> map cournot_quantity_step |> p
```

cov

`cov` — extended stdlib builtin. Alias for `covariance`.

```
my $result = cov $x
# or in a pipeline:
@list |> map cov |> p
```

cov2cor

`cov2cor` — convert covariance matrix to correlation matrix. Like R's cov2cor().

```
my $cor = cov2cor(cov_mat($data))
```

cov_mat

`cov_matrix` (alias `cov_mat`) — covariance matrix from observations.

```
my $S = cov_mat([[1,2],[3,4],[5,6]])
```

covar_pop

`covar_pop` — batch 70 builtin.

```
my $result = covar_pop $x
# or in a pipeline:
@list |> map covar_pop |> p
```

covar_samp

`covar_samp` — batch 70 builtin.

```
my $result = covar_samp $x
# or in a pipeline:
@list |> map covar_samp |> p
```

covariance

`covariance` — extended stdlib builtin.

```
my $result = covariance $x
# or in a pipeline:
@list |> map covariance |> p
```

covariance_matrix_pts

`covariance_matrix_pts` — batch 28 builtin.

```
my $result = covariance_matrix_pts $x
# or in a pipeline:
@list |> map covariance_matrix_pts |> p
```

covariant_derivative_step

`covariant_derivative_step` — batch 39 builtin.

```
my $result = covariant_derivative_step $x
# or in a pipeline:
@list |> map covariant_derivative_step |> p
```

cp_monatomic_ideal

`cp_monatomic_ideal` — batch 21 builtin.

```
my $result = cp_monatomic_ideal $x
# or in a pipeline:
@list |> map cp_monatomic_ideal |> p
```

cpat

`cpat` — extended stdlib builtin. Alias for `code_point_at`.

```
my $result = cpat $x
# or in a pipeline:
@list |> map cpat |> p
```

cpfx

`cpfx` — extended stdlib builtin. Alias for `common_prefix`.

```
my $result = cpfx $x
# or in a pipeline:
@list |> map cpfx |> p
```

cpl_w

`cpl_w` — batch 31 builtin.

```
my $result = cpl_w $x
# or in a pipeline:
@list |> map cpl_w |> p
```

cprod

`cprod` — python/ruby stdlib builtin. Alias for `cartesian_product`.

```
my $result = cprod $x
# or in a pipeline:
@list |> map cprod |> p
```

cprod_acc

`cprod_acc` — extended stdlib builtin. Alias for `cumprod`.

```
my $result = cprod_acc $x
# or in a pipeline:
@list |> map cprod_acc |> p
```

cpsgn

`cpsgn` — extended stdlib builtin. Alias for `copy_sign`.

```
my $result = cpsgn $x
# or in a pipeline:
@list |> map cpsgn |> p
```

cpu_count

`cpu_count` — more process / system builtin.

```
my $result = cpu_count $x
# or in a pipeline:
@list |> map cpu_count |> p
```

cpyw

`cpyw` — additional missing stdlib functions builtin. Alias for `copy_within`.

```
my $result = cpyw $x
# or in a pipeline:
@list |> map cpyw |> p
```

cr

`cr` — data / network builtin. Alias for `csv_read`.

```
my $result = cr $x
# or in a pipeline:
@list |> map cr |> p
```

cramer_lundberg_step

`cramer_lundberg_step` — batch 60 builtin.

```
my $result = cramer_lundberg_step $x
# or in a pipeline:
@list |> map cramer_lundberg_step |> p
```

cramer_rao_bound

`cramer_rao_bound` — batch 32 builtin.

```
my $result = cramer_rao_bound $x
# or in a pipeline:
@list |> map cramer_rao_bound |> p
```

crank_nicolson_ode

`crank_nicolson_ode` — batch 19 builtin.

```
my $result = crank_nicolson_ode $x
# or in a pipeline:
@list |> map crank_nicolson_ode |> p
```

crc16_ccitt

`crc16_ccitt` — batch 11 builtin.

```
my $result = crc16_ccitt $x
# or in a pipeline:
@list |> map crc16_ccitt |> p
```

credibility_buhlmann

`credibility_buhlmann` — batch 60 builtin.

```
my $result = credibility_buhlmann $x
# or in a pipeline:
@list |> map credibility_buhlmann |> p
```

crf_log_likelihood

`crf_log_likelihood` — batch 69 builtin.

```
my $result = crf_log_likelihood $x
# or in a pipeline:
@list |> map crf_log_likelihood |> p
```

crindex

`crindex STRING, SUBSTRING [, FROM]` — codepoint-indexed equivalent of Perl 5's `rindex`. Returns the last codepoint position where SUBSTRING occurs (at or before FROM, defaulting to end-of-string), or -1 on miss. Counterpart to `cindex`; pairs with `[]` slice operators and `len`.

```
my $path = "foo/bar/𝔻/baz.gz"
p rindex  $path, "/"        # 13 (bytes)
p crindex $path, "/"        # 11 (codepoints)
my $i = crindex $path, "/"
p $path[$i + 1 : len($path) - 1]   # "baz.gz"
```

critang

`critang` — physics formulas builtin. Alias for `critical_angle`.

```
my $result = critang $x
# or in a pipeline:
@list |> map critang |> p
```

critical_angle

`critical_angle` — physics formulas builtin.

```
my $result = critical_angle $x
# or in a pipeline:
@list |> map critical_angle |> p
```

critical_density

`critical_density` — batch 30 builtin.

```
my $result = critical_density $x
# or in a pipeline:
@list |> map critical_density |> p
```

critical_density_si

`critical_density_si` — batch 31 builtin.

```
my $result = critical_density_si $x
# or in a pipeline:
@list |> map critical_density_si |> p
```

cron_next_fire

`cron_next_fire` — batch 82 builtin.

```
my $result = cron_next_fire $x
# or in a pipeline:
@list |> map cron_next_fire |> p
```

cross_correlation

`cross_correlation(\@a, \@b)` (alias `xcorr`) — computes the cross-correlation of two signals. Measures similarity as a function of time-lag. Peak location indicates time delay between signals.

```
my $xcor = xcorr(\@signal1, \@signal2)
my $lag = max_index(@$xcor)   # find peak lag
```

cross_elasticity

`cross_elasticity` — batch 73 builtin.

```
my $result = cross_elasticity $x
# or in a pipeline:
@list |> map cross_elasticity |> p
```

cross_entropy

`cross_entropy` — math / numeric (uncategorized batch) builtin.

```
my $result = cross_entropy $x
# or in a pipeline:
@list |> map cross_entropy |> p
```

cross_polarization_amp

`cross_polarization_amp` — batch 39 builtin.

```
my $result = cross_polarization_amp $x
# or in a pipeline:
@list |> map cross_polarization_amp |> p
```

cross_product

`cross_product` — extended stdlib builtin.

```
my $result = cross_product $x
# or in a pipeline:
@list |> map cross_product |> p
```

crossp

`crossp` — extended stdlib builtin. Alias for `cross_product`.

```
my $result = crossp $x
# or in a pipeline:
@list |> map crossp |> p
```

crossprod

`crossprod` — cross product t(M) * M (R's crossprod). Efficiently computes M^T M without explicit transpose.

```
my $ata = crossprod([[1,2],[3,4]])  # [[10,14],[14,20]]
```

crr_american_call

`crr_american_call` — batch 20 builtin.

```
my $result = crr_american_call $x
# or in a pipeline:
@list |> map crr_american_call |> p
```

crr_american_put

`crr_american_put` — batch 20 builtin.

```
my $result = crr_american_put $x
# or in a pipeline:
@list |> map crr_american_put |> p
```

crustal_density_depth

`crustal_density_depth` — batch 65 builtin.

```
my $result = crustal_density_depth $x
# or in a pipeline:
@list |> map crustal_density_depth |> p
```

crystal_field_ligand

`crystal_field_ligand` — batch 63 builtin.

```
my $result = crystal_field_ligand $x
# or in a pipeline:
@list |> map crystal_field_ligand |> p
```

csc

`csc` returns the cosecant (1/sin) of an angle in radians.

```
p csc(1.5708)  # ≈ 1.0 (csc 90°)
```

csd_xy

`csd_xy` — batch 74 builtin.

```
my $result = csd_xy $x
# or in a pipeline:
@list |> map csd_xy |> p
```

csfx

`csfx` — extended stdlib builtin. Alias for `common_suffix`.

```
my $result = csfx $x
# or in a pipeline:
@list |> map csfx |> p
```

csiszar_phi_div

`csiszar_phi_div` — batch 40 builtin.

```
my $result = csiszar_phi_div $x
# or in a pipeline:
@list |> map csiszar_phi_div |> p
```

cspline

`cubic_spline` (aliases `cspline`, `spline`) performs natural cubic spline interpolation. Takes xs, ys, and a query point x.

```
p spline([0,1,2,3], [0,1,0,1], 1.5)  # smooth interpolation
```

csum

`csum` — extended stdlib builtin. Alias for `cumsum`.

```
my $result = csum $x
# or in a pipeline:
@list |> map csum |> p
```

cswap_gate

`cswap_gate` — batches 14-16 builtin. Alias for `fredkin_gate`.

```
my $result = cswap_gate $x
# or in a pipeline:
@list |> map cswap_gate |> p
```

ctime

`ctime` — file stat / path builtin. Alias for `file_ctime`.

```
my $result = ctime $x
# or in a pipeline:
@list |> map ctime |> p
```

ctxt

`ctxt` — extended stdlib builtin. Alias for `center_text`.

```
my $result = ctxt $x
# or in a pipeline:
@list |> map ctxt |> p
```

cuban_number

`cuban_number` — batch 10 builtin.

```
my $result = cuban_number $x
# or in a pipeline:
@list |> map cuban_number |> p
```

cube_fn

`cube_fn` — trig / math (batch 2) builtin.

```
my $result = cube_fn $x
# or in a pipeline:
@list |> map cube_fn |> p
```

cube_root

`cube_root` — math functions builtin.

```
my $result = cube_root $x
# or in a pipeline:
@list |> map cube_root |> p
```

cubes_seq

`cubes_seq` — sequences builtin.

```
my $result = cubes_seq $x
# or in a pipeline:
@list |> map cubes_seq |> p
```

cullen_number

`cullen_number` — batch 11 builtin.

```
my $result = cullen_number $x
# or in a pipeline:
@list |> map cullen_number |> p
```

cumipmt

`cumipmt` — batch 81 builtin.

```
my $result = cumipmt $x
# or in a pipeline:
@list |> map cumipmt |> p
```

cummax

`cummax` — cumulative maximum. Each element is the max of all elements up to that position.

```
p cummax([3,1,4,1,5,9])  # [3,3,4,4,5,9]
```

cummin

`cummin` — cumulative minimum.

```
p cummin([9,5,1,4,3])  # [9,5,1,1,1]
```

cumprinc

`cumprinc` — batch 81 builtin.

```
my $result = cumprinc $x
# or in a pipeline:
@list |> map cumprinc |> p
```

cumprod

`cumprod` — extended stdlib builtin.

```
my $result = cumprod $x
# or in a pipeline:
@list |> map cumprod |> p
```

cumsum

`cumsum` — extended stdlib builtin.

```
my $result = cumsum $x
# or in a pipeline:
@list |> map cumsum |> p
```

cumtrapz

`cumtrapz` cumulative trapezoidal integration. Returns running integral array.

```
my @y = (1, 2, 3, 4)
my @F = @{cumtrapz(\@y)}  # [0, 1.5, 4.0, 7.5]
```

cups_to_ml

`cups_to_ml` — file stat / path builtin.

```
my $result = cups_to_ml $input
```

current_divider

`current_divider` — batch 12 builtin.

```
my $result = current_divider $x
# or in a pipeline:
@list |> map current_divider |> p
```

curve25519_mul_simple

`curve25519_mul_simple` — batches 17-18 builtin.

```
my $result = curve25519_mul_simple $x
# or in a pipeline:
@list |> map curve25519_mul_simple |> p
```

cusip_check

`cusip_check` — batch 82 builtin.

```
my $result = cusip_check $x
# or in a pipeline:
@list |> map cusip_check |> p
```

cut

`cut` — bin continuous values into intervals. Returns integer bin indices. Like R's cut().

```
p cut([1.5, 3.2, 7.8], [0, 2, 5, 10])  # [1, 2, 3]
```

cutree

`cutree` — cut a dendrogram into k clusters. Takes merge list from hclust and k. Returns cluster assignments. Like R's cutree().

```
my @clusters = @{cutree(hclust(dist_mat($data)), 3)}
```

cutting_plane_step

`cutting_plane_step` — batch 41 builtin.

```
my $result = cutting_plane_step $x
# or in a pipeline:
@list |> map cutting_plane_step |> p
```

cv

`cv` — math / numeric (uncategorized batch) builtin. Alias for `coeff_of_variation`.

```
my $result = cv $x
# or in a pipeline:
@list |> map cv |> p
```

cv_monatomic_ideal

`cv_monatomic_ideal` — batch 21 builtin.

```
my $result = cv_monatomic_ideal $x
# or in a pipeline:
@list |> map cv_monatomic_ideal |> p
```

cvar_historical

`cvar_historical` — batches 17-18 builtin.

```
my $result = cvar_historical $x
# or in a pipeline:
@list |> map cvar_historical |> p
```

cw

`cw` — data / network builtin. Alias for `csv_write`.

```
my $result = cw $x
# or in a pipeline:
@list |> map cw |> p
```

cwd

`cwd` — more process / system builtin.

```
my $result = cwd $x
# or in a pipeline:
@list |> map cwd |> p
```

cwt_morlet

`cwt_morlet` — batch 74 builtin.

```
my $result = cwt_morlet $x
# or in a pipeline:
@list |> map cwt_morlet |> p
```

cyan

`cyan` — color / ansi builtin. Alias for `red`.

```
my $result = cyan $x
# or in a pipeline:
@list |> map cyan |> p
```

cyan_bold

`cyan_bold` — color / ansi builtin. Alias for `red`.

```
my $result = cyan_bold $x
# or in a pipeline:
@list |> map cyan_bold |> p
```

cyber_banner

`cyber_banner` (alias `neon_banner`) — large neon block-letter banner with gradient coloring and border. Args: text.

```
p cyber_banner("STRYKE")
p cyber_banner("HACK THE PLANET")
```

cyber_circuit

`cyber_circuit` — circuit board pattern with traces, intersections, and glowing nodes. Args: [width, height, seed].

```
p cyber_circuit()             # 60x20 default
p cyber_circuit(80, 30, 42)   # custom
```

cyber_city

`cyber_city` — procedural neon cityscape with buildings, windows, stars, and antennas. Args: [width, height, seed]. Output: ANSI-colored string for terminal.

```
p cyber_city()              # 80x24 default
p cyber_city(120, 40, 99)   # custom size and seed
```

cyber_eye

`cyber_eye` — cyberpunk all-seeing eye motif with layered glow. Args: [size]. Size: "small" (default) or "large".

```
p cyber_eye()          # small eye
p cyber_eye("large")   # large eye
```

cyber_glitch

`cyber_glitch` (alias `glitch_text`) — glitch-distort text with ANSI corruption, screen tears, and neon color bleeding. Args: text [, intensity 1-10].

```
p cyber_glitch("SYSTEM BREACH", 7)
p cyber_glitch("hello world")
```

cyber_grid

`cyber_grid` — retro perspective grid with vanishing point and neon glow (Tron/synthwave style). Args: [width, height].

```
p cyber_grid()           # 80x24 default
p cyber_grid(120, 30)    # wider grid
```

cyber_rain

`cyber_rain` (alias `matrix_rain`) — matrix-style digital rain with Japanese katakana and green phosphor glow. Args: [width, height, seed].

```
p cyber_rain()              # 80x24 default
p cyber_rain(120, 40, 42)   # custom
```

cyber_skull

`cyber_skull` — neon skull ASCII art with glitch effects. Args: [size]. Size: "small" (default) or "large".

```
p cyber_skull()          # small skull
p cyber_skull("large")   # large skull
```

cyc

`cyc` — algebraic match builtin. Alias for `cycle`.

```
my $result = cyc $x
# or in a pipeline:
@list |> map cyc |> p
```

cycle

`cycle` — algebraic match builtin.

```
my $result = cycle $x
# or in a pipeline:
@list |> map cycle |> p
```

cycle_n

`cycle_n` — collection (batch 2) builtin. Alias for `repeat_list`.

```
my $result = cycle_n $x
# or in a pipeline:
@list |> map cycle_n |> p
```

cyclic_homology_step

`cyclic_homology_step` — batch 37 builtin.

```
my $result = cyclic_homology_step $x
# or in a pipeline:
@list |> map cyclic_homology_step |> p
```

cylinder_surface

`cylinder_surface` — batch 28 builtin.

```
my $result = cylinder_surface $x
# or in a pipeline:
@list |> map cylinder_surface |> p
```

cylinder_volume

`cylinder_volume` — geometry / physics builtin.

```
my $result = cylinder_volume $x
# or in a pipeline:
@list |> map cylinder_volume |> p
```

cylvol

`cylvol` — geometry / physics builtin. Alias for `cylinder_volume`.

```
my $result = cylvol $x
# or in a pipeline:
@list |> map cylvol |> p
```

d

`d` — filesystem extensions builtin.

```
my $result = d $x
# or in a pipeline:
@list |> map d |> p
```

d2comp

`d2comp` — misc / utility builtin. Alias for `degrees_to_compass`.

```
my $result = d2comp $x
# or in a pipeline:
@list |> map d2comp |> p
```

d2r

`d2r` — trivial numeric / predicate builtins builtin. Alias for `deg_to_rad`.

```
my $result = d2r $x
# or in a pipeline:
@list |> map d2r |> p
```

dB_power

`dB_power` — batch 12 builtin.

```
my $result = dB_power $x
# or in a pipeline:
@list |> map dB_power |> p
```

dB_voltage

`dB_voltage` — batch 12 builtin.

```
my $result = dB_voltage $x
# or in a pipeline:
@list |> map dB_voltage |> p
```

dagum_pdf

dagum_pdf — batches 17-18 builtin.

```
my $result = dagum_pdf $x
# or in a pipeline:
@list |> map dagum_pdf |> p
```

daitch_mokotoff

daitch_mokotoff — batch 69 builtin.

```
my $result = daitch_mokotoff $x
# or in a pipeline:
@list |> map daitch_mokotoff |> p
```

dalys_disability_weight

dalys_disability_weight — batch 61 builtin.

```
my $result = dalys_disability_weight $x
# or in a pipeline:
@list |> map dalys_disability_weight |> p
```

damerau_levenshtein

damerau_levenshtein — batch 33 builtin.

```
my $result = damerau_levenshtein $x
# or in a pipeline:
@list |> map damerau_levenshtein |> p
```

dantzig_wolfe_step

dantzig_wolfe_step — batch 41 builtin.

```
my $result = dantzig_wolfe_step $x
# or in a pipeline:
@list |> map dantzig_wolfe_step |> p
```

darboux_frame_step

darboux_frame_step — batch 39 builtin.

```
my $result = darboux_frame_step $x
# or in a pipeline:
@list |> map darboux_frame_step |> p
```

dasum

dasum — batch 66 builtin.

```
my $result = dasum $x
# or in a pipeline:
@list |> map dasum |> p
```

data_processing_inequality

data_processing_inequality — batch 40 builtin.

```
my $result = data_processing_inequality $x
# or in a pipeline:
@list |> map data_processing_inequality |> p
```

date_round_iso

date_round_iso — batch 82 builtin.

```
my $result = date_round_iso $x
# or in a pipeline:
@list |> map date_round_iso |> p
```

dawsn

dawsn — batch 72 builtin.

```
my $result = dawsn $x
# or in a pipeline:
@list |> map dawsn |> p
```

dawson_function

dawson_function — batch 29 builtin.

```
my $result = dawson_function $x
# or in a pipeline:
@list |> map dawson_function |> p
```

daxpy

daxpy — batch 66 builtin.

```
my $result = daxpy $x
# or in a pipeline:
@list |> map daxpy |> p
```

day_of_week_iso

day_of_week_iso — batch 9 builtin.

```
my $result = day_of_week_iso $x
# or in a pipeline:
@list |> map day_of_week_iso |> p
```

day_of_week_zeller

day_of_week_zeller — batch 52 builtin.

```
my $result = day_of_week_zeller $x
# or in a pipeline:
@list |> map day_of_week_zeller |> p
```

day_of_year

day_of_year — extended stdlib batch 3 builtin.

```
my $result = day_of_year $x
# or in a pipeline:
@list |> map day_of_year |> p
```

days_between

`days_between` — batch 9 builtin.

```
my $result = days_between $x
# or in a pipeline:
@list |> map days_between |> p
```

days_in_month

`days_in_month` — date helpers builtin.

```
my $result = days_in_month $x
# or in a pipeline:
@list |> map days_in_month |> p
```

days_in_month_fn

`days_in_month_fn` — extended stdlib batch 3 builtin.

```
my $result = days_in_month_fn $x
# or in a pipeline:
@list |> map days_in_month_fn |> p
```

days_in_year

`days_in_year` — batch 13 builtin.

```
my $result = days_in_year $x
# or in a pipeline:
@list |> map days_in_year |> p
```

days_to_hours

`days_to_hours` — unit conversions builtin.

```
my $result = days_to_hours $input
```

days_to_seconds

`days_to_seconds` — unit conversions builtin.

```
my $result = days_to_seconds $input
```

daysinmo

`daysinmo` — extended stdlib batch 3 builtin. Alias for `days_in_month_fn`.

```
my $result = daysinmo $x
# or in a pipeline:
@list |> map daysinmo |> p
```

db

`db` — physics formulas builtin. Alias for `decibel_ratio`.

```
my $result = db $x
# or in a pipeline:
@list |> map db |> p
```

db4_coeffs

db4_coeffs — batches 14-16 builtin.

```
my $result = db4_coeffs $x
# or in a pipeline:
@list |> map db4_coeffs |> p
```

db4_wavelet_step

db4_wavelet_step — batch 40 builtin.

```
my $result = db4_wavelet_step $x
# or in a pipeline:
@list |> map db4_wavelet_step |> p
```

db6_coeffs

db6_coeffs — batches 14-16 builtin.

```
my $result = db6_coeffs $x
# or in a pipeline:
@list |> map db6_coeffs |> p
```

db_anti_entropy_pull_step

db_anti_entropy_pull_step — batch 48 builtin.

```
my $result = db_anti_entropy_pull_step $x
# or in a pipeline:
@list |> map db_anti_entropy_pull_step |> p
```

db_anti_entropy_step

db_anti_entropy_step — batch 48 builtin.

```
my $result = db_anti_entropy_step $x
# or in a pipeline:
@list |> map db_anti_entropy_step |> p
```

db_arc_cache_score

db_arc_cache_score — batch 48 builtin.

```
my $result = db_arc_cache_score $x
# or in a pipeline:
@list |> map db_arc_cache_score |> p
```

db_async_commit_check

db_async_commit_check — batch 48 builtin.

```
my $result = db_async_commit_check $x
# or in a pipeline:
@list |> map db_async_commit_check |> p
```

db_avalanche_query_step

db_avalanche_query_step — batch 48 builtin.

```
my $result = db_avalanche_query_step $x
# or in a pipeline:
@list |> map db_avalanche_query_step |> p
```

db_b_tree_merge

db_b_tree_merge — batch 48 builtin.

```
my $result = db_b_tree_merge $x
# or in a pipeline:
@list |> map db_b_tree_merge |> p
```

db_b_tree_split

db_b_tree_split — batch 48 builtin.

```
my $result = db_b_tree_split $x
# or in a pipeline:
@list |> map db_b_tree_split |> p
```

db_block_cache_hit_rate

db_block_cache_hit_rate — batch 48 builtin.

```
my $result = db_block_cache_hit_rate $x
# or in a pipeline:
@list |> map db_block_cache_hit_rate |> p
```

db_bloom_filter_bit_index

db_bloom_filter_bit_index — batch 48 builtin.

```
my $result = db_bloom_filter_bit_index $x
# or in a pipeline:
@list |> map db_bloom_filter_bit_index |> p
```

db_buffer_pool_score

db_buffer_pool_score — batch 48 builtin.

```
my $result = db_buffer_pool_score $x
# or in a pipeline:
@list |> map db_buffer_pool_score |> p
```

db_byzantine_quorum_size

db_byzantine_quorum_size — batch 48 builtin.

```
my $result = db_byzantine_quorum_size $x
# or in a pipeline:
@list |> map db_byzantine_quorum_size |> p
```

db_causal_consistency_check

db_causal_consistency_check — batch 48 builtin.

```
my $result = db_causal_consistency_check $x
# or in a pipeline:
@list |> map db_causal_consistency_check |> p
```

db_chao_estimator_step

db_chao_estimator_step — batch 48 builtin.

```
my $result = db_chao_estimator_step $x
# or in a pipeline:
@list |> map db_chao_estimator_step |> p
```

db_chord_finger_index

db_chord_finger_index — batch 48 builtin.

```
my $result = db_chord_finger_index $x
# or in a pipeline:
@list |> map db_chord_finger_index |> p
```

db_chubby_lease_step

db_chubby_lease_step — batch 48 builtin.

```
my $result = db_chubby_lease_step $x
# or in a pipeline:
@list |> map db_chubby_lease_step |> p
```

db_circuit_breaker_step

db_circuit_breaker_step — batch 48 builtin.

```
my $result = db_circuit_breaker_step $x
# or in a pipeline:
@list |> map db_circuit_breaker_step |> p
```

db_clock_cache_hand

db_clock_cache_hand — batch 48 builtin.

```
my $result = db_clock_cache_hand $x
# or in a pipeline:
@list |> map db_clock_cache_hand |> p
```

db_clock_skew_estimate

db_clock_skew_estimate — batch 48 builtin.

```
my $result = db_clock_skew_estimate $x
# or in a pipeline:
@list |> map db_clock_skew_estimate |> p
```

db_compaction_score

db_compaction_score — batch 48 builtin.

```
my $result = db_compaction_score $x
# or in a pipeline:
@list |> map db_compaction_score |> p
```

db_consensus_quorum_size

db_consensus_quorum_size — batch 48 builtin.

```
my $result = db_consensus_quorum_size $x
# or in a pipeline:
@list |> map db_consensus_quorum_size |> p
```

db_consistent_hash_index

db_consistent_hash_index — batch 48 builtin.

```
my $result = db_consistent_hash_index $x
# or in a pipeline:
@list |> map db_consistent_hash_index |> p
```

db_consistent_lookup_id

db_consistent_lookup_id — batch 48 builtin.

```
my $result = db_consistent_lookup_id $x
# or in a pipeline:
@list |> map db_consistent_lookup_id |> p
```

db_count_min_sketch_bin

db_count_min_sketch_bin — batch 48 builtin.

```
my $result = db_count_min_sketch_bin $x
# or in a pipeline:
@list |> map db_count_min_sketch_bin |> p
```

db_crdt_g_counter_merge

db_crdt_g_counter_merge — batch 48 builtin.

```
my $result = db_crdt_g_counter_merge $x
# or in a pipeline:
@list |> map db_crdt_g_counter_merge |> p
```

db_crdt_lww_register_merge

db_crdt_lww_register_merge — batch 48 builtin.

```
my $result = db_crdt_lww_register_merge $x
# or in a pipeline:
@list |> map db_crdt_lww_register_merge |> p
```

db_crdt_pn_counter_merge

db_crdt_pn_counter_merge — batch 48 builtin.

```
my $result = db_crdt_pn_counter_merge $x
# or in a pipeline:
@list |> map db_crdt_pn_counter_merge |> p
```

db_crdt_set_or_merge

db_crdt_set_or_merge — batch 48 builtin.

```
my $result = db_crdt_set_or_merge $x
# or in a pipeline:
@list |> map db_crdt_set_or_merge |> p
```

db_cuckoo_filter_fingerprint

db_cuckoo_filter_fingerprint — batch 48 builtin.

```
my $result = db_cuckoo_filter_fingerprint $x
# or in a pipeline:
@list |> map db_cuckoo_filter_fingerprint |> p
```

db_dd_sketch_bin

db_dd_sketch_bin — batch 48 builtin.

```
my $result = db_dd_sketch_bin $x
# or in a pipeline:
@list |> map db_dd_sketch_bin |> p
```

db_dht_replicate_factor

db_dht_replicate_factor — batch 48 builtin.

```
my $result = db_dht_replicate_factor $x
# or in a pipeline:
@list |> map db_dht_replicate_factor |> p
```

db_distinct_estimate_hll

db_distinct_estimate_hll — batch 48 builtin.

```
my $result = db_distinct_estimate_hll $x
# or in a pipeline:
@list |> map db_distinct_estimate_hll |> p
```

db_distinct_estimate_lpc

db_distinct_estimate_lpc — batch 48 builtin.

```
my $result = db_distinct_estimate_lpc $x
# or in a pipeline:
@list |> map db_distinct_estimate_lpc |> p
```

db_eventual_consistency_check

db_eventual_consistency_check — batch 48 builtin.

```
my $result = db_eventual_consistency_check $x
# or in a pipeline:
@list |> map db_eventual_consistency_check |> p
```

db_freshness_score

db_freshness_score — batch 48 builtin.

```
my $result = db_freshness_score $x
# or in a pipeline:
@list |> map db_freshness_score |> p
```

db_gossip_fanout_step

db_gossip_fanout_step — batch 48 builtin.

```
my $result = db_gossip_fanout_step $x
# or in a pipeline:
@list |> map db_gossip_fanout_step |> p
```

db_group_commit_count

db_group_commit_count — batch 48 builtin.

```
my $result = db_group_commit_count $x
# or in a pipeline:
@list |> map db_group_commit_count |> p
```

db_hash_join_cost

db_hash_join_cost — batch 48 builtin.

```
my $result = db_hash_join_cost $x
# or in a pipeline:
@list |> map db_hash_join_cost |> p
```

db_hinted_handoff_step

db_hinted_handoff_step — batch 48 builtin.

```
my $result = db_hinted_handoff_step $x
# or in a pipeline:
@list |> map db_hinted_handoff_step |> p
```

db_histogram_bucket_index

db_histogram_bucket_index — batch 48 builtin.

```
my $result = db_histogram_bucket_index $x
# or in a pipeline:
@list |> map db_histogram_bucket_index |> p
```

db_honey_badger_step

db_honey_badger_step — batch 48 builtin.

```
my $result = db_honey_badger_step $x
# or in a pipeline:
@list |> map db_honey_badger_step |> p
```

db_hybrid_logical_clock

db_hybrid_logical_clock — batch 48 builtin.

```
my $result = db_hybrid_logical_clock $x
# or in a pipeline:
@list |> map db_hybrid_logical_clock |> p
```

db_hyperloglog_register_max

db_hyperloglog_register_max — batch 48 builtin.

```
my $result = db_hyperloglog_register_max $x
# or in a pipeline:
@list |> map db_hyperloglog_register_max |> p
```

db_index_scan_cost

db_index_scan_cost — batch 48 builtin.

```
my $result = db_index_scan_cost $x
# or in a pipeline:
@list |> map db_index_scan_cost |> p
```

db_index_seek_cost

db_index_seek_cost — batch 48 builtin.

```
my $result = db_index_seek_cost $x
# or in a pipeline:
@list |> map db_index_seek_cost |> p
```

db_jaccard_minhash_estimate

db_jaccard_minhash_estimate — batch 48 builtin.

```
my $result = db_jaccard_minhash_estimate $x
# or in a pipeline:
@list |> map db_jaccard_minhash_estimate |> p
```

db_join_selectivity_step

db_join_selectivity_step — batch 48 builtin.

```
my $result = db_join_selectivity_step $x
# or in a pipeline:
@list |> map db_join_selectivity_step |> p
```

db_jump_hash_bucket

db_jump_hash_bucket — batch 48 builtin.

```
my $result = db_jump_hash_bucket $x
# or in a pipeline:
@list |> map db_jump_hash_bucket |> p
```

db_kademlia_xor_distance

db_kademlia_xor_distance — batch 48 builtin.

```
my $result = db_kademlia_xor_distance $x
# or in a pipeline:
@list |> map db_kademlia_xor_distance |> p
```

db_kll_quantile_step

db_kll_quantile_step — batch 48 builtin.

```
my $result = db_kll_quantile_step $x
# or in a pipeline:
@list |> map db_kll_quantile_step |> p
```

db_lamport_timestamp

db_lamport_timestamp — batch 48 builtin.

```
my $result = db_lamport_timestamp $x
# or in a pipeline:
@list |> map db_lamport_timestamp |> p
```

db_leaky_bucket_step

db_leaky_bucket_step — batch 48 builtin.

```
my $result = db_leaky_bucket_step $x
# or in a pipeline:
@list |> map db_leaky_bucket_step |> p
```

db_levelled_compaction_step

db_levelled_compaction_step — batch 48 builtin.

```
my $result = db_levelled_compaction_step $x
# or in a pipeline:
@list |> map db_levelled_compaction_step |> p
```

db_lfu_cache_decay

db_lfu_cache_decay — batch 48 builtin.

```
my $result = db_lfu_cache_decay $x
# or in a pipeline:
@list |> map db_lfu_cache_decay |> p
```

db_linearizability_check

`db_linearizability_check` — batch 48 builtin.

```
my $result = db_linearizability_check $x
# or in a pipeline:
@list |> map db_linearizability_check |> p
```

db_logical_clock_step

`db_logical_clock_step` — batch 48 builtin.

```
my $result = db_logical_clock_step $x
# or in a pipeline:
@list |> map db_logical_clock_step |> p
```

db_lru_cache_eviction_age

`db_lru_cache_eviction_age` — batch 48 builtin.

```
my $result = db_lru_cache_eviction_age $x
# or in a pipeline:
@list |> map db_lru_cache_eviction_age |> p
```

db_lsm_compaction_step

`db_lsm_compaction_step` — batch 48 builtin.

```
my $result = db_lsm_compaction_step $x
# or in a pipeline:
@list |> map db_lsm_compaction_step |> p
```

db_maglev_hash_step

`db_maglev_hash_step` — batch 48 builtin.

```
my $result = db_maglev_hash_step $x
# or in a pipeline:
@list |> map db_maglev_hash_step |> p
```

db_merge_join_cost

`db_merge_join_cost` — batch 48 builtin.

```
my $result = db_merge_join_cost $x
# or in a pipeline:
@list |> map db_merge_join_cost |> p
```

db_merkle_node_hash

`db_merkle_node_hash` — batch 48 builtin.

```
my $result = db_merkle_node_hash $x
# or in a pipeline:
@list |> map db_merkle_node_hash |> p
```

db_merkle_path_verify

`db_merkle_path_verify` — batch 48 builtin.

```
my $result = db_merkle_path_verify $x
# or in a pipeline:
@list |> map db_merkle_path_verify |> p
```

db_min_hash_value

db_min_hash_value — batch 48 builtin.

```
my $result = db_min_hash_value $x
# or in a pipeline:
@list |> map db_min_hash_value |> p
```

db_nested_loop_cost

db_nested_loop_cost — batch 48 builtin.

```
my $result = db_nested_loop_cost $x
# or in a pipeline:
@list |> map db_nested_loop_cost |> p
```

db_page_cache_eviction_age

db_page_cache_eviction_age — batch 48 builtin.

```
my $result = db_page_cache_eviction_age $x
# or in a pipeline:
@list |> map db_page_cache_eviction_age |> p
```

db_partition_failure_check

db_partition_failure_check — batch 48 builtin.

```
my $result = db_partition_failure_check $x
# or in a pipeline:
@list |> map db_partition_failure_check |> p
```

db_partitions_for_n

db_partitions_for_n — batch 48 builtin.

```
my $result = db_partitions_for_n $x
# or in a pipeline:
@list |> map db_partitions_for_n |> p
```

db_pastry_routing_step

db_pastry_routing_step — batch 48 builtin.

```
my $result = db_pastry_routing_step $x
# or in a pipeline:
@list |> map db_pastry_routing_step |> p
```

db_paxos_propose_id

db_paxos_propose_id — batch 48 builtin.

```
my $result = db_paxos_propose_id $x
# or in a pipeline:
@list |> map db_paxos_propose_id |> p
```

db_pbft_view_change

db_pbft_view_change — batch 48 builtin.

```
my $result = db_pbft_view_change $x
# or in a pipeline:
@list |> map db_pbft_view_change |> p
```

db_power

`db_power` — batch 12 builtin. Alias for `dB_power`.

```
my $result = db_power $x
# or in a pipeline:
@list |> map db_power |> p
```

db_quantile_estimate_p99

`db_quantile_estimate_p99` — batch 48 builtin.

```
my $result = db_quantile_estimate_p99 $x
# or in a pipeline:
@list |> map db_quantile_estimate_p99 |> p
```

db_query_cardinality

`db_query_cardinality` — batch 48 builtin.

```
my $result = db_query_cardinality $x
# or in a pipeline:
@list |> map db_query_cardinality |> p
```

db_query_plan_cost_step

`db_query_plan_cost_step` — batch 48 builtin.

```
my $result = db_query_plan_cost_step $x
# or in a pipeline:
@list |> map db_query_plan_cost_step |> p
```

db_quorum_intersection_check

`db_quorum_intersection_check` — batch 48 builtin.

```
my $result = db_quorum_intersection_check $x
# or in a pipeline:
@list |> map db_quorum_intersection_check |> p
```

db_quotient_filter_canonical

`db_quotient_filter_canonical` — batch 48 builtin.

```
my $result = db_quotient_filter_canonical $x
# or in a pipeline:
@list |> map db_quotient_filter_canonical |> p
```

db_raft_log_match_check

`db_raft_log_match_check` — batch 48 builtin.

```
my $result = db_raft_log_match_check $x
# or in a pipeline:
@list |> map db_raft_log_match_check |> p
```

db_raft_term_advance

`db_raft_term_advance` — batch 48 builtin.

```
my $result = db_raft_term_advance $x
# or in a pipeline:
@list |> map db_raft_term_advance |> p
```

db_read_amplification

db_read_amplification — batch 48 builtin.

```
my $result = db_read_amplification $x
# or in a pipeline:
@list |> map db_read_amplification |> p
```

db_read_repair_step

db_read_repair_step — batch 48 builtin.

```
my $result = db_read_repair_step $x
# or in a pipeline:
@list |> map db_read_repair_step |> p
```

db_rendezvous_hash_score

db_rendezvous_hash_score — batch 48 builtin.

```
my $result = db_rendezvous_hash_score $x
# or in a pipeline:
@list |> map db_rendezvous_hash_score |> p
```

db_replica_lag_threshold

db_replica_lag_threshold — batch 48 builtin.

```
my $result = db_replica_lag_threshold $x
# or in a pipeline:
@list |> map db_replica_lag_threshold |> p
```

db_replication_lag_step

db_replication_lag_step — batch 48 builtin.

```
my $result = db_replication_lag_step $x
# or in a pipeline:
@list |> map db_replication_lag_step |> p
```

db_reservoir_sample_index

db_reservoir_sample_index — batch 48 builtin.

```
my $result = db_reservoir_sample_index $x
# or in a pipeline:
@list |> map db_reservoir_sample_index |> p
```

db_seq_scan_cost

db_seq_scan_cost — batch 48 builtin.

```
my $result = db_seq_scan_cost $x
# or in a pipeline:
@list |> map db_seq_scan_cost |> p
```

db_simhash_bit

db_simhash_bit — batch 48 builtin.

```
my $result = db_simhash_bit $x
# or in a pipeline:
@list |> map db_simhash_bit |> p
```

db_size_tiered_compaction

db_size_tiered_compaction — batch 48 builtin.

```
my $result = db_size_tiered_compaction $x
# or in a pipeline:
@list |> map db_size_tiered_compaction |> p
```

db_skiplist_height_pick

db_skiplist_height_pick — batch 48 builtin.

```
my $result = db_skiplist_height_pick $x
# or in a pipeline:
@list |> map db_skiplist_height_pick |> p
```

db_sort_cost_estimate

db_sort_cost_estimate — batch 48 builtin.

```
my $result = db_sort_cost_estimate $x
# or in a pipeline:
@list |> map db_sort_cost_estimate |> p
```

db_space_amplification

db_space_amplification — batch 48 builtin.

```
my $result = db_space_amplification $x
# or in a pipeline:
@list |> map db_space_amplification |> p
```

db_split_brain_check

db_split_brain_check — batch 48 builtin.

```
my $result = db_split_brain_check $x
# or in a pipeline:
@list |> map db_split_brain_check |> p
```

db_strong_consistency_check

db_strong_consistency_check — batch 48 builtin.

```
my $result = db_strong_consistency_check $x
# or in a pipeline:
@list |> map db_strong_consistency_check |> p
```

db_synchronous_commit_check

db_synchronous_commit_check — batch 48 builtin.

```
my $result = db_synchronous_commit_check $x
# or in a pipeline:
@list |> map db_synchronous_commit_check |> p
```

db_t_digest_centroid

db_t_digest_centroid — batch 48 builtin.

```
my $result = db_t_digest_centroid $x
# or in a pipeline:
@list |> map db_t_digest_centroid |> p
```

db_three_phase_commit_step

db_three_phase_commit_step — batch 48 builtin.

```
my $result = db_three_phase_commit_step $x
# or in a pipeline:
@list |> map db_three_phase_commit_step |> p
```

db_throttle_token_step

db_throttle_token_step — batch 48 builtin.

```
my $result = db_throttle_token_step $x
# or in a pipeline:
@list |> map db_throttle_token_step |> p
```

db_tinylfu_admit_score

db_tinylfu_admit_score — batch 48 builtin.

```
my $result = db_tinylfu_admit_score $x
# or in a pipeline:
@list |> map db_tinylfu_admit_score |> p
```

db_to_amp

db_to_amp — batch 30 builtin.

```
my $result = db_to_amp $input
```

db_token_bucket_step

db_token_bucket_step — batch 48 builtin.

```
my $result = db_token_bucket_step $x
# or in a pipeline:
@list |> map db_token_bucket_step |> p
```

db_two_phase_commit_step

db_two_phase_commit_step — batch 48 builtin.

```
my $result = db_two_phase_commit_step $x
# or in a pipeline:
@list |> map db_two_phase_commit_step |> p
```

db_universal_compaction_step

db_universal_compaction_step — batch 48 builtin.

```
my $result = db_universal_compaction_step $x
# or in a pipeline:
@list |> map db_universal_compaction_step |> p
```

db_vector_clock_merge

db_vector_clock_merge — batch 48 builtin.

```
my $result = db_vector_clock_merge $x
# or in a pipeline:
@list |> map db_vector_clock_merge |> p
```

db_voltage

`db_voltage` — batch 12 builtin. Alias for `dB_voltage`.

```
my $result = db_voltage $x
# or in a pipeline:
@list |> map db_voltage |> p
```

db_w_tinylfu_freq

`db_w_tinylfu_freq` — batch 48 builtin.

```
my $result = db_w_tinylfu_freq $x
# or in a pipeline:
@list |> map db_w_tinylfu_freq |> p
```

db_wal_fsync_cost

`db_wal_fsync_cost` — batch 48 builtin.

```
my $result = db_wal_fsync_cost $x
# or in a pipeline:
@list |> map db_wal_fsync_cost |> p
```

db_write_amplification

`db_write_amplification` — batch 48 builtin.

```
my $result = db_write_amplification $x
# or in a pipeline:
@list |> map db_write_amplification |> p
```

db_zab_epoch_step

`db_zab_epoch_step` — batch 48 builtin.

```
my $result = db_zab_epoch_step $x
# or in a pipeline:
@list |> map db_zab_epoch_step |> p
```

dbinom

`dbinom` — binomial PMF. P(X = k) for Binom(n, p). Args: k, n, p.

```
p dbinom(5, 10, 0.5)   # P(exactly 5 heads in 10 flips)
```

dbmclose

`dbmclose` — ipc builtin.

```
my $result = dbmclose $x
# or in a pipeline:
@list |> map dbmclose |> p
```

dbmopen

`dbmopen` — ipc builtin.

```
my $result = dbmopen $x
# or in a pipeline:
@list |> map dbmopen |> p
```

dbsize

`dbsize` — batch 71 builtin.

```
my $result = dbsize $x
# or in a pipeline:
@list |> map dbsize |> p
```

dchisq

`dchisq` — batch 53 builtin.

```
my $result = dchisq $x
# or in a pipeline:
@list |> map dchisq |> p
```

dct

`dct` computes the Type-II Discrete Cosine Transform of a signal. Used in JPEG, MP3, and speech processing.

```
my @coeffs = @{dct([1,2,3,4])}
```

dde_euler_step

`dde_euler_step` — batch 34 builtin.

```
my $result = dde_euler_step $x
# or in a pipeline:
@list |> map dde_euler_step |> p
```

ddot

`ddot` — batch 66 builtin.

```
my $result = ddot $x
# or in a pipeline:
@list |> map ddot |> p
```

ddpg_critic_loss_step

`ddpg_critic_loss_step` — batch 40 builtin.

```
my $result = ddpg_critic_loss_step $x
# or in a pipeline:
@list |> map ddpg_critic_loss_step |> p
```

ddrot

`ddrot` — batch 66 builtin.

```
my $result = ddrot $x
# or in a pipeline:
@list |> map ddrot |> p
```

ddt

`ddt` — extended stdlib builtin. Alias for `dedent`.

```
my $result = ddt $x
# or in a pipeline:
@list |> map ddt |> p
```

de_broglie

`de_broglie` — batch 21 builtin.

```
my $result = de_broglie $x
# or in a pipeline:
@list |> map de_broglie |> p
```

de_broglie_wavelength

`de_broglie_wavelength($mass, $velocity)` (alias `debroglie`) — quantum wavelength $\lambda = h/(mv)$. Defaults to electron mass.

```
p debroglie(9.109e-31, 1e6)   # ~7.3e-10 m (electron at 1 km/s)
```

de_broglie_wavelength_kg

`de_broglie_wavelength_kg` — batch 12 builtin.

```
my $result = de_broglie_wavelength_kg $x
# or in a pipeline:
@list |> map de_broglie_wavelength_kg |> p
```

de_rham_zero

`de_rham_zero` — batch 37 builtin.

```
my $result = de_rham_zero $x
# or in a pipeline:
@list |> map de_rham_zero |> p
```

de_sitter_metric_step

`de_sitter_metric_step` — batch 39 builtin.

```
my $result = de_sitter_metric_step $x
# or in a pipeline:
@list |> map de_sitter_metric_step |> p
```

de_sitter_radius_step

`de_sitter_radius_step` — batch 39 builtin.

```
my $result = de_sitter_radius_step $x
# or in a pipeline:
@list |> map de_sitter_radius_step |> p
```

deadweight_loss

`deadweight_loss` — batch 73 builtin.

```
my $result = deadweight_loss $x
# or in a pipeline:
@list |> map deadweight_loss |> p
```

deal_n_k

`deal_n_k` — batch 54 builtin.

```
my $result = deal_n_k $x
# or in a pipeline:
@list |> map deal_n_k |> p
```

debug_object

`debug_object` — batch 71 builtin.

```
my $result = debug_object $x
# or in a pipeline:
@list |> map debug_object |> p
```

debug_val

`debug_val` — functional combinators builtin.

```
my $result = debug_val $x
# or in a pipeline:
@list |> map debug_val |> p
```

debye_huckel

`debye_huckel` — batch 21 builtin.

```
my $result = debye_huckel $x
# or in a pipeline:
@list |> map debye_huckel |> p
```

debye_huckel_activity

`debye_huckel_activity` — batch 38 builtin.

```
my $result = debye_huckel_activity $x
# or in a pipeline:
@list |> map debye_huckel_activity |> p
```

debye_huckel_screening_factor

`debye_huckel_screening_factor` — batch 38 builtin.

```
my $result = debye_huckel_screening_factor $x
# or in a pipeline:
@list |> map debye_huckel_screening_factor |> p
```

debye_length_electrolyte

`debye_length_electrolyte` — batch 38 builtin.

```
my $result = debye_length_electrolyte $x
# or in a pipeline:
@list |> map debye_length_electrolyte |> p
```

dec

`dec` — pipeline / string helpers builtin.

```
my $result = dec $x
# or in a pipeline:
@list |> map dec |> p
```

dec_each

`dec_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = dec_each $x
# or in a pipeline:
@list |> map dec_each |> p
```

deceleration_q

`deceleration_q` — batch 31 builtin.

```
my $result = deceleration_q $x
# or in a pipeline:
@list |> map deceleration_q |> p
```

decibel_ratio

`decibel_ratio` — physics formulas builtin.

```
my $result = decibel_ratio $x
# or in a pipeline:
@list |> map decibel_ratio |> p
```

decimate_step

`decimate_step` — batch 74 builtin.

```
my $result = decimate_step $x
# or in a pipeline:
@list |> map decimate_step |> p
```

decode_base

`decode_base` — batch 54 builtin.

```
my $result = decode_base $x
# or in a pipeline:
@list |> map decode_base |> p
```

decoherence_time

`decoherence_time` — batch 32 builtin.

```
my $result = decoherence_time $x
# or in a pipeline:
@list |> map decoherence_time |> p
```

deconvolve_step

`deconvolve_step` — batch 74 builtin.

```
my $result = deconvolve_step $x
# or in a pipeline:
@list |> map deconvolve_step |> p
```

decr

`decr` — batch 71 builtin.

```
my $result = decr $x
# or in a pipeline:
@list |> map decr |> p
```

decrby

`decrby` — batch 71 builtin.

```
my $result = decrby $x
# or in a pipeline:
@list |> map decrby |> p
```

dedent

`dedent` — extended stdlib builtin.

```
my $result = dedent $x
# or in a pipeline:
@list |> map dedent |> p
```

dedent_text

`dedent_text` — string processing (uncategorized batch) builtin.

```
my $result = dedent_text $x
# or in a pipeline:
@list |> map dedent_text |> p
```

deep_q_target

`deep_q_target` — batch 40 builtin.

```
my $result = deep_q_target $x
# or in a pipeline:
@list |> map deep_q_target |> p
```

default_to

`default_to` — functional combinators builtin.

```
my $result = default_to $x
# or in a pipeline:
@list |> map default_to |> p
```

defaultdict

`defaultdict` — data structure helpers builtin.

```
my $result = defaultdict $x
# or in a pipeline:
@list |> map defaultdict |> p
```

deferred_acceptance

`deferred_acceptance` — batch 73 builtin.

```
my $result = deferred_acceptance $x
# or in a pipeline:
@list |> map deferred_acceptance |> p
```

deferred_acceptance_step

`deferred_acceptance_step` — batch 43 builtin.

```
my $result = deferred_acceptance_step $x
# or in a pipeline:
@list |> map deferred_acceptance_step |> p
```

deficient_numbers

`deficient_numbers` — number theory / primes builtin.

```
my $result = deficient_numbers $x
# or in a pipeline:
@list |> map deficient_numbers |> p
```

defined_count

`defined_count` — counters (batch 4) builtin.

```
my $result = defined_count $x
# or in a pipeline:
@list |> map defined_count |> p
```

definums

`definums` — number theory / primes builtin. Alias for `deficient_numbers`.

```
my $result = definums $x
# or in a pipeline:
@list |> map definums |> p
```

deflate_dynamic_huffman

`deflate_dynamic_huffman` — batch 62 builtin.

```
my $result = deflate_dynamic_huffman $x
# or in a pipeline:
@list |> map deflate_dynamic_huffman |> p
```

deflate_encode_huffman

`deflate_encode_huffman` — batch 81 builtin.

```
my $result = deflate_encode_huffman $x
# or in a pipeline:
@list |> map deflate_encode_huffman |> p
```

deflate_huffman_lit

`deflate_huffman_lit` — batch 40 builtin.

```
my $result = deflate_huffman_lit $x
# or in a pipeline:
@list |> map deflate_huffman_lit |> p
```

deflate_static_block

`deflate_static_block` — batch 62 builtin.

```
my $result = deflate_static_block $x
# or in a pipeline:
@list |> map deflate_static_block |> p
```

deg_to_rad

`deg_to_rad` — trivial numeric / predicate builtins builtin.

```
my $result = deg_to_rad $input
```

degrees

`degrees` — trig / math (batch 2) builtin.

```
my $result = degrees $x
# or in a pipeline:
@list |> map degrees |> p
```

degrees_to_compass

`degrees_to_compass` — misc / utility builtin.

```
my $result = degrees_to_compass $input
```

dela

`dela` — go/general functional utilities builtin. Alias for `delete_at`.

```
my $result = dela $x
# or in a pipeline:
@list |> map dela |> p
```

delby

`delby` — additional missing stdlib functions builtin. Alias for `delete_by`.

```
my $result = delby $x
# or in a pipeline:
@list |> map delby |> p
```

delete_at

`delete_at` — go/general functional utilities builtin.

```
my $result = delete_at $x
# or in a pipeline:
@list |> map delete_at |> p
```

delete_by

`delete_by` — additional missing stdlib functions builtin.

```
my $result = delete_by $x
# or in a pipeline:
@list |> map delete_by |> p
```

delete_first

`delete_first` — additional missing stdlib functions builtin.

```
my $result = delete_first $x
# or in a pipeline:
@list |> map delete_first |> p
```

delfst

`delfst` — additional missing stdlib functions builtin. Alias for `delete_first`.

```
my $result = delfst $x
# or in a pipeline:
@list |> map delfst |> p
```

delta_complex_count

`delta_complex_count` — batch 37 builtin.

```
my $result = delta_complex_count $x
# or in a pipeline:
@list |> map delta_complex_count |> p
```

delta_e

`delta_e` — batch 9 builtin. Alias for `delta_e_76`.

```
my $result = delta_e $x
# or in a pipeline:
@list |> map delta_e |> p
```

delta_e2000

`delta_e2000` — batch 82 builtin.

```
my $result = delta_e2000 $x
# or in a pipeline:
@list |> map delta_e2000 |> p
```

delta_e76

`delta_e76` — batch 82 builtin.

```
my $result = delta_e76 $x
# or in a pipeline:
@list |> map delta_e76 |> p
```

delta_e94

`delta_e94` — batch 82 builtin.

```
my $result = delta_e94 $x
# or in a pipeline:
@list |> map delta_e94 |> p
```

delta_e_76

`delta_e_76` — batch 9 builtin.

```
my $result = delta_e_76 $x
# or in a pipeline:
@list |> map delta_e_76 |> p
```

delta_e_94

`delta_e_94` — batch 30 builtin.

```
my $result = delta_e_94 $x
# or in a pipeline:
@list |> map delta_e_94 |> p
```

demorse

`demorse` — encoding / phonetics builtin. Alias for `morse_decode`.

```
my $result = demorse $x
# or in a pipeline:
@list |> map demorse |> p
```

denavit_hartenberg_h

`denavit_hartenberg_h` — batch 59 builtin.

```
my $result = denavit_hartenberg_h $x
# or in a pipeline:
@list |> map denavit_hartenberg_h |> p
```

dense_rank

`dense_rank` — extended stdlib builtin.

```
my $result = dense_rank $x
# or in a pipeline:
@list |> map dense_rank |> p
```

density

`density` — kernel density estimation with Gaussian kernel and Silverman bandwidth. Returns [[x_grid], [density_values]]. Like R's density().

```
my ($x, $y) = @{density([1,2,2,3,3,3,4,4,5])}
# $x is grid of 512 points, $y is estimated density
```

density_altitude_full

`density_altitude_full` — batch 42 builtin.

```
my $result = density_altitude_full $x
# or in a pipeline:
@list |> map density_altitude_full |> p
```

density_altitude_m

`density_altitude_m` — batch 12 builtin.

```
my $result = density_altitude_m $x
# or in a pipeline:
@list |> map density_altitude_m |> p
```

density_matrix_purity_q

`density_matrix_purity_q` — batch 80 builtin.

```
my $result = density_matrix_purity_q $x
# or in a pipeline:
@list |> map density_matrix_purity_q |> p
```

depdbl

`depdbl` — math / numeric (uncategorized batch) builtin. Alias for `depreciation_double`.

```
my $result = depdbl $x
# or in a pipeline:
@list |> map depdbl |> p
```

dependency_parse_eisner

`dependency_parse_eisner` — batch 69 builtin.

```
my $result = dependency_parse_eisner $x
# or in a pipeline:
@list |> map dependency_parse_eisner |> p
```

dephasing_channel

`dephasing_channel` — batches 14-16 builtin.

```
my $result = dephasing_channel $x
# or in a pipeline:
@list |> map dephasing_channel |> p
```

deplin

`deplin` — math / numeric (uncategorized batch) builtin. Alias for `depreciation_linear`.

```
my $result = deplin $x
# or in a pipeline:
@list |> map deplin |> p
```

depolarizing_channel

`depolarizing_channel` — batches 14-16 builtin.

```
my $result = depolarizing_channel $x
# or in a pipeline:
@list |> map depolarizing_channel |> p
```

depolarizing_density_2x2

`depolarizing_density_2x2` — batch 32 builtin.

```
my $result = depolarizing_density_2x2 $x
# or in a pipeline:
@list |> map depolarizing_density_2x2 |> p
```

depreciation_double

`depreciation_double` — math / numeric (uncategorized batch) builtin.

```
my $result = depreciation_double $x
# or in a pipeline:
@list |> map depreciation_double |> p
```

depreciation_linear

`depreciation_linear` — math / numeric (uncategorized batch) builtin.

```
my $result = depreciation_linear $x
# or in a pipeline:
@list |> map depreciation_linear |> p
```

derangements

`derangements` counts the number of derangements (subfactorial !n) — permutations with no fixed points.

```
p derangements(4)   # 9
p derangements(5)   # 44
```

deroman

`deroman` — roman numerals builtin. Alias for `roman_to_int`.

```
my $result = deroman $x
# or in a pipeline:
@list |> map deroman |> p
```

describe

`describe(@data)` — returns a hashref with comprehensive descriptive statistics: count, mean, std, min, 25%, 50%, 75%, max. Similar to pandas DataFrame.describe().

```
my $stats = describe(1:100)
p $stats->{mean}    # 50.5
p $stats->{std}     # ~29.0
p $stats->{"50%"}   # median
```

destination_point

`destination_point` — batch 58 builtin.

```
my $result = destination_point $x
# or in a pipeline:
@list |> map destination_point |> p
```

det

`matrix_det_general` (aliases `mdetg`, `det`) computes the determinant of any NxN matrix via LU decomposition.

```
p det([[1,2,3],[4,5,6],[7,8,0]])  # 27
```

detect

`detect` — functional / iterator builtin.

```
my $result = detect $x
# or in a pipeline:
@list |> map detect |> p
```

deterministic_prime

`deterministic_prime` — batch 26 builtin.

```
my $result = deterministic_prime $x
# or in a pipeline:
@list |> map deterministic_prime |> p
```

detrend_linear

`detrend_linear` — batch 74 builtin.

```
my $result = detrend_linear $x
# or in a pipeline:
@list |> map detrend_linear |> p
```

detrended_fluct_alpha

`detrended_fluct_alpha` — batch 42 builtin.

```
my $result = detrended_fluct_alpha $x
# or in a pipeline:
@list |> map detrended_fluct_alpha |> p
```

dew_point

`dew_point` — physics formulas builtin.

```
my $result = dew_point $x
# or in a pipeline:
@list |> map dew_point |> p
```

dew_point_magnus

`dew_point_magnus` — batch 12 builtin.

```
my $result = dew_point_magnus $x
# or in a pipeline:
@list |> map dew_point_magnus |> p
```

dewpoint_temperature_full

`dewpoint_temperature_full` — batch 42 builtin.

```
my $result = dewpoint_temperature_full $x
# or in a pipeline:
@list |> map dewpoint_temperature_full |> p
```

df

`df` — data / network builtin. Alias for `dataframe`.

```
my $result = df $x
# or in a pipeline:
@list |> map df |> p
```

df_aggregate

`df_aggregate` — batch 76 builtin.

```
my $result = df_aggregate $x
# or in a pipeline:
@list |> map df_aggregate |> p
```

df_apply

`df_apply` — batch 76 builtin.

```
my $result = df_apply $x
# or in a pipeline:
@list |> map df_apply |> p
```

df_clip

`df_clip` — batch 76 builtin.

```
my $result = df_clip $x
# or in a pipeline:
@list |> map df_clip |> p
```

df_concat

`df_concat` — batch 76 builtin.

```
my $result = df_concat $x
# or in a pipeline:
@list |> map df_concat |> p
```

df_corr

`df_corr` — batch 76 builtin.

```
my $result = df_corr $x
# or in a pipeline:
@list |> map df_corr |> p
```

df_corrwith

`df_corrwith` — batch 76 builtin.

```
my $result = df_corrwith $x
# or in a pipeline:
@list |> map df_corrwith |> p
```

df_cov

`df_cov` — batch 76 builtin.

```
my $result = df_cov $x
# or in a pipeline:
@list |> map df_cov |> p
```

df_crosstab

`df_crosstab` — batch 76 builtin.

```
my $result = df_crosstab $x
# or in a pipeline:
@list |> map df_crosstab |> p
```

df_describe

`df_describe` — batch 76 builtin.

```
my $result = df_describe $x
# or in a pipeline:
@list |> map df_describe |> p
```

df_diff

`df_diff` — batch 76 builtin.

```
my $result = df_diff $x
# or in a pipeline:
@list |> map df_diff |> p
```

df_dist

`df_dist` — batch 53 builtin.

```
my $result = df_dist $x
# or in a pipeline:
@list |> map df_dist |> p
```

df_drop_duplicates

`df_drop_duplicates` — batch 76 builtin.

```
my $result = df_drop_duplicates $x
# or in a pipeline:
@list |> map df_drop_duplicates |> p
```

df_dropna

`df_dropna` — batch 76 builtin.

```
my $result = df_dropna $x
# or in a pipeline:
@list |> map df_dropna |> p
```

df_duplicated

`df_duplicated` — batch 76 builtin.

```
my $result = df_duplicated $x
# or in a pipeline:
@list |> map df_duplicated |> p
```

df_eval

`df_eval` — batch 76 builtin.

```
my $result = df_eval $x
# or in a pipeline:
@list |> map df_eval |> p
```

df_ewm

`df_ewm` — batch 76 builtin.

```
my $result = df_ewm $x
# or in a pipeline:
@list |> map df_ewm |> p
```

df_expanding

`df_expanding` — batch 76 builtin.

```
my $result = df_expanding $x
# or in a pipeline:
@list |> map df_expanding |> p
```

df_explode

`df_explode` — batch 76 builtin.

```
my $result = df_explode $x
# or in a pipeline:
@list |> map df_explode |> p
```

df_fillna

`df_fillna` — batch 76 builtin.

```
my $result = df_fillna $x
# or in a pipeline:
@list |> map df_fillna |> p
```

df_filter

`df_filter` — batch 76 builtin.

```
my $result = df_filter $x
# or in a pipeline:
@list |> map df_filter |> p
```

df_get_dummies

`df_get_dummies` — batch 76 builtin.

```
my $result = df_get_dummies $x
# or in a pipeline:
@list |> map df_get_dummies |> p
```

df_groupby

`df_groupby` — batch 76 builtin.

```
my $result = df_groupby $x
# or in a pipeline:
@list |> map df_groupby |> p
```

df_idxmax

`df_idxmax` — batch 76 builtin.

```
my $result = df_idxmax $x
# or in a pipeline:
@list |> map df_idxmax |> p
```

df_idxmin

`df_idxmin` — batch 76 builtin.

```
my $result = df_idxmin $x
# or in a pipeline:
@list |> map df_idxmin |> p
```

df_interpolate

`df_interpolate` — batch 76 builtin.

```
my $result = df_interpolate $x
# or in a pipeline:
@list |> map df_interpolate |> p
```

df_isnull

`df_isnull` — batch 76 builtin.

```
my $result = df_isnull $x
# or in a pipeline:
@list |> map df_isnull |> p
```

df_join

`df_join` — batch 76 builtin.

```
my $result = df_join $x
# or in a pipeline:
@list |> map df_join |> p
```

df_kurtosis

df_kurtosis — batch 76 builtin.

```
my $result = df_kurtosis $x
# or in a pipeline:
@list |> map df_kurtosis |> p
```

df_mad

df_mad — batch 76 builtin.

```
my $result = df_mad $x
# or in a pipeline:
@list |> map df_mad |> p
```

df_melt

df_melt — batch 76 builtin.

```
my $result = df_melt $x
# or in a pipeline:
@list |> map df_melt |> p
```

df_merge

df_merge — batch 76 builtin.

```
my $result = df_merge $x
# or in a pipeline:
@list |> map df_merge |> p
```

df_nlargest

df_nlargest — batch 76 builtin.

```
my $result = df_nlargest $x
# or in a pipeline:
@list |> map df_nlargest |> p
```

df_notnull

df_notnull — batch 76 builtin.

```
my $result = df_notnull $x
# or in a pipeline:
@list |> map df_notnull |> p
```

df_nsmallest

df_nsmallest — batch 76 builtin.

```
my $result = df_nsmallest $x
# or in a pipeline:
@list |> map df_nsmallest |> p
```

df_pct_change

df_pct_change — batch 76 builtin.

```
my $result = df_pct_change $x
# or in a pipeline:
@list |> map df_pct_change |> p
```

df_pivot

`df_pivot` — batch 76 builtin.

```
my $result = df_pivot $x
# or in a pipeline:
@list |> map df_pivot |> p
```

df_pivot_table

`df_pivot_table` — batch 76 builtin.

```
my $result = df_pivot_table $x
# or in a pipeline:
@list |> map df_pivot_table |> p
```

df_quantile

`df_quantile` — batch 76 builtin.

```
my $result = df_quantile $x
# or in a pipeline:
@list |> map df_quantile |> p
```

df_query

`df_query` — batch 76 builtin.

```
my $result = df_query $x
# or in a pipeline:
@list |> map df_query |> p
```

df_rank

`df_rank` — batch 76 builtin.

```
my $result = df_rank $x
# or in a pipeline:
@list |> map df_rank |> p
```

df_replace

`df_replace` — batch 76 builtin.

```
my $result = df_replace $x
# or in a pipeline:
@list |> map df_replace |> p
```

df_resample

`df_resample` — batch 76 builtin.

```
my $result = df_resample $x
# or in a pipeline:
@list |> map df_resample |> p
```

df_reset_index

`df_reset_index` — batch 76 builtin.

```
my $result = df_reset_index $x
# or in a pipeline:
@list |> map df_reset_index |> p
```

df_rolling

`df_rolling` — batch 76 builtin.

```
my $result = df_rolling $x
# or in a pipeline:
@list |> map df_rolling |> p
```

df_round

`df_round` — batch 76 builtin.

```
my $result = df_round $x
# or in a pipeline:
@list |> map df_round |> p
```

df_sample

`df_sample` — batch 76 builtin.

```
my $result = df_sample $x
# or in a pipeline:
@list |> map df_sample |> p
```

df_sem

`df_sem` — batch 76 builtin.

```
my $result = df_sem $x
# or in a pipeline:
@list |> map df_sem |> p
```

df_set_index

`df_set_index` — batch 76 builtin.

```
my $result = df_set_index $x
# or in a pipeline:
@list |> map df_set_index |> p
```

df_shift

`df_shift` — batch 76 builtin.

```
my $result = df_shift $x
# or in a pipeline:
@list |> map df_shift |> p
```

df_skew

`df_skew` — batch 76 builtin.

```
my $result = df_skew $x
# or in a pipeline:
@list |> map df_skew |> p
```

df_sort_values

`df_sort_values` — batch 76 builtin.

```
my $result = df_sort_values $x
# or in a pipeline:
@list |> map df_sort_values |> p
```

df_stack

df_stack — batch 76 builtin.

```
my $result = df_stack $x
# or in a pipeline:
@list |> map df_stack |> p
```

df_to_datetime

df_to_datetime — batch 76 builtin.

```
my $result = df_to_datetime $input
```

df_to_numeric

df_to_numeric — batch 76 builtin.

```
my $result = df_to_numeric $input
```

df_to_timedelta

df_to_timedelta — batch 76 builtin.

```
my $result = df_to_timedelta $input
```

df_transform

df_transform — batch 76 builtin.

```
my $result = df_transform $x
# or in a pipeline:
@list |> map df_transform |> p
```

df_unstack

df_unstack — batch 76 builtin.

```
my $result = df_unstack $x
# or in a pipeline:
@list |> map df_unstack |> p
```

df_value_counts

df_value_counts — batch 76 builtin.

```
my $result = df_value_counts $x
# or in a pipeline:
@list |> map df_value_counts |> p
```

dfa_minimize_hopcroft

dfa_minimize_hopcroft — batch 68 builtin.

```
my $result = dfa_minimize_hopcroft $x
# or in a pipeline:
@list |> map dfa_minimize_hopcroft |> p
```

dfa_simulate_step

`dfa_simulate_step` — batch 68 builtin.

```
my $result = dfa_simulate_step $x
# or in a pipeline:
@list |> map dfa_simulate_step |> p
```

dfact

`dfact` — math formulas builtin. Alias for `double_factorial`.

```
my $result = dfact $x
# or in a pipeline:
@list |> map dfact |> p
```

dfs

`dfs` — extended stdlib builtin. Alias for `dfs_traverse`.

```
my $result = dfs $x
# or in a pipeline:
@list |> map dfs |> p
```

dfs_postorder_done

`dfs_postorder_done` — batch 75 builtin.

```
my $result = dfs_postorder_done $x
# or in a pipeline:
@list |> map dfs_postorder_done |> p
```

dfs_preorder

`dfs_preorder` — batch 13 builtin.

```
my $result = dfs_preorder $x
# or in a pipeline:
@list |> map dfs_preorder |> p
```

dfs_traverse

`dfs_traverse` — extended stdlib builtin.

```
my $result = dfs_traverse $x
# or in a pipeline:
@list |> map dfs_traverse |> p
```

dft

`dft(\@signal)` — computes the Discrete Fourier Transform. Returns arrayref of complex numbers as [re, im] pairs. O(n²) reference implementation; for large n, use FFT libraries.

```
my @signal = map { sin(2 * 3.14159 * _ / 64) } 0:63
my $spectrum = dft(\@signal)
```

dg_from_k

`dg_from_k` — batch 21 builtin.

```
my $result = dg_from_k $x
# or in a pipeline:
@list |> map dg_from_k |> p
```

dgbsv

dgbsv — batch 66 builtin.

```
my $result = dgbsv $x
# or in a pipeline:
@list |> map dgbsv |> p
```

dgels

dgels — batch 66 builtin.

```
my $result = dgels $x
# or in a pipeline:
@list |> map dgels |> p
```

dgelsd

dgelsd — batch 66 builtin.

```
my $result = dgelsd $x
# or in a pipeline:
@list |> map dgelsd |> p
```

dgemm

dgemm — batch 66 builtin.

```
my $result = dgemm $x
# or in a pipeline:
@list |> map dgemm |> p
```

dgemm3m

dgemm3m — batch 66 builtin.

```
my $result = dgemm3m $x
# or in a pipeline:
@list |> map dgemm3m |> p
```

dgemv

dgemv — batch 66 builtin.

```
my $result = dgemv $x
# or in a pipeline:
@list |> map dgemv |> p
```

dgeom

dgeom — geometric PMF P(X=k). Args: k, prob.

```
p dgeom(3, 0.5)  # P(first success on 4th trial)
```

dgeqrf

dgeqrf — batch 66 builtin.

```
my $result = dgeqrf $x
# or in a pipeline:
@list |> map dgeqrf |> p
```

dgerqf

dgerqf — batch 66 builtin.

```
my $result = dgerqf $x
# or in a pipeline:
@list |> map dgerqf |> p
```

dgesv

dgesv — batch 66 builtin.

```
my $result = dgesv $x
# or in a pipeline:
@list |> map dgesv |> p
```

dgesvd

dgesvd — batch 66 builtin.

```
my $result = dgesvd $x
# or in a pipeline:
@list |> map dgesvd |> p
```

dgetrf

dgetrf — batch 66 builtin.

```
my $result = dgetrf $x
# or in a pipeline:
@list |> map dgetrf |> p
```

dh_secret

dh_secret — batch 35 builtin.

```
my $result = dh_secret $x
# or in a pipeline:
@list |> map dh_secret |> p
```

dh_shared

dh_shared — batch 26 builtin.

```
my $result = dh_shared $x
# or in a pipeline:
@list |> map dh_shared |> p
```

dhyper

dhyper — hypergeometric PMF. Args: k, m (white), n (black), nn (draws).

```
p dhyper(2, 5, 5, 3)  # P(2 white balls in 3 draws from 5W+5B)
```

diag

`diag` — extended stdlib builtin. Alias for `diagonal`.

```
my $result = diag $x
# or in a pipeline:
@list |> map diag |> p
```

diagonal

`diagonal` — extended stdlib builtin.

```
my $result = diagonal $x
# or in a pipeline:
@list |> map diagonal |> p
```

diagonal_dominance_q

`diagonal_dominance_q` — batches 17-18 builtin.

```
my $result = diagonal_dominance_q $x
# or in a pipeline:
@list |> map diagonal_dominance_q |> p
```

diameter_bfs

`diameter_bfs` — batch 24 builtin.

```
my $result = diameter_bfs $x
# or in a pipeline:
@list |> map diameter_bfs |> p
```

diameter_unweighted

`diameter_unweighted` — batches 14-16 builtin.

```
my $result = diameter_unweighted $x
# or in a pipeline:
@list |> map diameter_unweighted |> p
```

dice_coefficient

`dice_coefficient` — extended stdlib batch 3 builtin.

```
my $result = dice_coefficient $x
# or in a pipeline:
@list |> map dice_coefficient |> p
```

dice_roll

`dice_roll` — random builtin.

```
my $result = dice_roll $x
# or in a pipeline:
@list |> map dice_roll |> p
```

dicecoef

`dicecoef` — extended stdlib batch 3 builtin. Alias for `dice_coefficient`.

```
my $result = dicecoef $x
# or in a pipeline:
@list |> map dicecoef |> p
```

dickey_fuller_critical

`dickey_fuller_critical` — batch 36 builtin.

```
my $result = dickey_fuller_critical $x
# or in a pipeline:
@list |> map dickey_fuller_critical |> p
```

dictator_game_share

`dictator_game_share` — batch 43 builtin.

```
my $result = dictator_game_share $x
# or in a pipeline:
@list |> map dictator_game_share |> p
```

did_estimator

`did_estimator` — batch 73 builtin.

```
my $result = did_estimator $x
# or in a pipeline:
@list |> map did_estimator |> p
```

die_if

`die_if` — conversion / utility (batch 4) builtin.

```
my $result = die_if $x
# or in a pipeline:
@list |> map die_if |> p
```

die_unless

`die_unless` — conversion / utility (batch 4) builtin.

```
my $result = die_unless $x
# or in a pipeline:
@list |> map die_unless |> p
```

diesel_efficiency

`diesel_efficiency` — batch 12 builtin.

```
my $result = diesel_efficiency $x
# or in a pipeline:
@list |> map diesel_efficiency |> p
```

diff

`diff` — extended stdlib builtin.

```
my $result = diff $x
# or in a pipeline:
@list |> map diff |> p
```

diff_array

`numerical_diff` (aliases `numdiff`, `diff_array`) computes the numerical first derivative via central differences. Returns an array.

```
my @y = map { _ ** 2 } 0:10
my @dy = @{numdiff(\@y)}  # ≈ [0, 2, 4, 6, ...]
```

diff_coeff_aqueous_estimate

diff_coeff_aqueous_estimate — batch 38 builtin.

```
my $result = diff_coeff_aqueous_estimate $x
# or in a pipeline:
@list |> map diff_coeff_aqueous_estimate |> p
```

diff_days

diff_days — extended stdlib builtin.

```
my $result = diff_days $x
# or in a pipeline:
@list |> map diff_days |> p
```

diff_hours

diff_hours — extended stdlib builtin.

```
my $result = diff_hours $x
# or in a pipeline:
@list |> map diff_hours |> p
```

diff_in_diff

diff_in_diff — batch 73 builtin.

```
my $result = diff_in_diff $x
# or in a pipeline:
@list |> map diff_in_diff |> p
```

diff_lag

diff_lag (alias diff_ts) — lagged differences. Args: vec [, lag, differences]. Like R's diff().

```
p diff_lag([1,3,6,10])       # [2,3,4] (lag=1)
p diff_lag([1,3,6,10], 2)    # [5,7] (lag=2)
p diff_lag([1,3,6,10], 1, 2) # [1,1] (second differences)
```

diffd

diffd — extended stdlib builtin. Alias for diff_days.

```
my $result = diffd $x
# or in a pipeline:
@list |> map diffd |> p
```

diffh

diffh — extended stdlib builtin. Alias for diff_hours.

```
my $result = diffh $x
# or in a pipeline:
@list |> map diffh |> p
```

diffraction_grating_angle

diffraction_grating_angle — batch 23 builtin.

```
my $result = diffraction_grating_angle $x
# or in a pipeline:
@list |> map diffraction_grating_angle |> p
```

diffuse_layer_thickness

`diffuse_layer_thickness` — batch 38 builtin.

```
my $result = diffuse_layer_thickness $x
# or in a pipeline:
@list |> map diffuse_layer_thickness |> p
```

diffusion_layer_thickness

`diffusion_layer_thickness` — batch 38 builtin.

```
my $result = diffusion_layer_thickness $x
# or in a pipeline:
@list |> map diffusion_layer_thickness |> p
```

diffusion_stability

`diffusion_stability` — batch 34 builtin.

```
my $result = diffusion_stability $x
# or in a pipeline:
@list |> map diffusion_stability |> p
```

dig

`dig` — python/ruby stdlib builtin.

```
my $result = dig $x
# or in a pipeline:
@list |> map dig |> p
```

digest_auth_quote

`digest_auth_quote` — batch 81 builtin.

```
my $result = digest_auth_quote $x
# or in a pipeline:
@list |> map digest_auth_quote |> p
```

digital_call

`digital_call` — batch 20 builtin.

```
my $result = digital_call $x
# or in a pipeline:
@list |> map digital_call |> p
```

digital_root

`digital_root` — extended stdlib batch 3 builtin.

```
my $result = digital_root $x
# or in a pipeline:
@list |> map digital_root |> p
```

digitize

`digitize` — batch 72 builtin.

```
my $result = digitize $x
# or in a pipeline:
@list |> map digitize |> p
```

digits_of

`digits_of` — list helpers (batch 4) builtin.

```
my $result = digits_of $x
# or in a pipeline:
@list |> map digits_of |> p
```

digroot

`digroot` — extended stdlib batch 3 builtin. Alias for `digital_root`.

```
my $result = digroot $x
# or in a pipeline:
@list |> map digroot |> p
```

dijkstra

`dijkstra` (alias `shortest_path`) computes shortest paths from a source node. Graph is a hash of {node => [[neighbor, weight], ...]}. Returns {node => distance}.

```
my $g = {A => [["B",1],["C",4]], B => [["C",2]], C => []}
my $d = dijkstra($g, "A")  # {A=>0, B=>1, C=>3}
```

dijkstra_relax

`dijkstra_relax` — batch 75 builtin.

```
my $result = dijkstra_relax $x
# or in a pipeline:
@list |> map dijkstra_relax |> p
```

dilution_v2

`dilution_v2` — batch 21 builtin.

```
my $result = dilution_v2 $x
# or in a pipeline:
@list |> map dilution_v2 |> p
```

dim

`dim` — color / ansi builtin. Alias for `red`.

```
my $result = dim $x
# or in a pipeline:
@list |> map dim |> p
```

dinic_blocking_flow

`dinic_blocking_flow` — batch 41 builtin.

```
my $result = dinic_blocking_flow $x
# or in a pipeline:
@list |> map dinic_blocking_flow |> p
```

dinic_step

`dinic_step` — batch 75 builtin.

```
my $result = dinic_step $x
# or in a pipeline:
@list |> map dinic_step |> p
```

diopter

`diopter` — physics formulas builtin. Alias for `lens_power`.

```
my $result = diopter $x
# or in a pipeline:
@list |> map diopter |> p
```

direct_age_adjusted

`direct_age_adjusted` — batch 61 builtin.

```
my $result = direct_age_adjusted $x
# or in a pipeline:
@list |> map direct_age_adjusted |> p
```

dirs

`dirs` — filesystem extensions builtin.

```
my $result = dirs $x
# or in a pipeline:
@list |> map dirs |> p
```

dirty_price

`dirty_price` — batch 57 builtin.

```
my $result = dirty_price $x
# or in a pipeline:
@list |> map dirty_price |> p
```

disc_excel

`disc_excel` — batch 81 builtin.

```
my $result = disc_excel $x
# or in a pipeline:
@list |> map disc_excel |> p
```

discount

`discount` — math / numeric (uncategorized batch) builtin.

```
my $result = discount $x
# or in a pipeline:
@list |> map discount |> p
```

discount_amount

`discount_amount` — physics formulas builtin.

```
my $result = discount_amount $x
# or in a pipeline:
@list |> map discount_amount |> p
```

discount_continuous

`discount_continuous` — batch 20 builtin.

```
my $result = discount_continuous $x
# or in a pipeline:
@list |> map discount_continuous |> p
```

discount_factor_step

`discount_factor_step` — batch 43 builtin.

```
my $result = discount_factor_step $x
# or in a pipeline:
@list |> map discount_factor_step |> p
```

discounted_payback

`discounted_payback($initial, $rate, @cashflows)` — computes payback period using discounted cash flows. Accounts for time value of money.

```
p discounted_payback(1000, 0.1, 400, 400, 400, 400)
```

discrete_haar_step

`discrete_haar_step` — batch 40 builtin.

```
my $result = discrete_haar_step $x
# or in a pipeline:
@list |> map discrete_haar_step |> p
```

disk_avail

`disk_avail PATH` — available disk space in bytes for non-root users. Uses `f_bavail` from `statvfs`, which excludes blocks reserved for the superuser.

```
p format_bytes(disk_avail)              # 76.3 GiB
gauge(1 - disk_avail / disk_total) |> p  # usage gauge
```

disk_free

`disk_free PATH` — free disk space in bytes (superuser view). Uses `f_bfree` from `statvfs`.

```
p format_bytes(disk_free)               # 76.3 GiB
p format_bytes(disk_free("/tmp"))
```

disk_total

`disk_total PATH` — total disk space in bytes for the filesystem containing PATH (default `/`). Uses `statvfs` on unix. Returns `undef` on unsupported platforms.

```
p format_bytes(disk_total)              # 1.8 TiB
p format_bytes(disk_total("/home"))      # specific mount
```

disk_used

`disk_used PATH` — used disk space in bytes (total - free).

```
p format_bytes(disk_used)               # 1.7 TiB
my $pct = int(disk_used / disk_total * 100)
p "$pct% disk used"
```

dissoc

`dissoc` — algebraic match builtin.

```
my $result = dissoc $x
# or in a pipeline:
@list |> map dissoc |> p
```

dist

`dist` — extended stdlib builtin. Alias for `distance`.

```
my $result = dist $x
# or in a pipeline:
@list |> map dist |> p
```

dist_mat

`dist_matrix` (alias `dist_mat`) — pairwise distance matrix. Supports 'euclidean' (default), 'manhattan', 'maximum'. Like R's dist().

```
my $D = dist_mat([[0,0],[1,0],[0,1]])  # 3x3 distance matrix
my $D2 = dist_mat([[0,0],[1,1]], "manhattan")  # Manhattan
```

dist_point_line_2d

`dist_point_line_2d` — batch 28 builtin.

```
my $result = dist_point_line_2d $x
# or in a pipeline:
@list |> map dist_point_line_2d |> p
```

dist_point_plane_3d

`dist_point_plane_3d` — batch 28 builtin.

```
my $result = dist_point_plane_3d $x
# or in a pipeline:
@list |> map dist_point_plane_3d |> p
```

distance

`distance` — extended stdlib builtin.

```
my $result = distance $x
# or in a pipeline:
@list |> map distance |> p
```

distb

`distb` — additional missing stdlib functions builtin. Alias for `distinct_by`.

```
my $result = distb $x
# or in a pipeline:
@list |> map distb |> p
```

distinct_by

`distinct_by` — additional missing stdlib functions builtin.

```
my $result = distinct_by $x
# or in a pipeline:
@list |> map distinct_by |> p
```

distinct_count

`distinct_count` — collection (batch 2) builtin.

```
my $result = distinct_count $x
# or in a pipeline:
@list |> map distinct_count |> p
```

divergence_omega_step

`divergence_omega_step` — batch 42 builtin.

```
my $result = divergence_omega_step $x
# or in a pipeline:
@list |> map divergence_omega_step |> p
```

divisible_goods_proportional

`divisible_goods_proportional` — batch 43 builtin.

```
my $result = divisible_goods_proportional $x
# or in a pipeline:
@list |> map divisible_goods_proportional |> p
```

divisors

`divisors` — math / numeric (uncategorized batch) builtin.

```
my $result = divisors $x
# or in a pipeline:
@list |> map divisors |> p
```

divmod

`divmod` — python/ruby stdlib builtin.

```
my $result = divmod $x
# or in a pipeline:
@list |> map divmod |> p
```

divs

`divs` — math / numeric (uncategorized batch) builtin. Alias for `divisors`.

```
my $result = divs $x
# or in a pipeline:
@list |> map divs |> p
```

djb2

`djb2` — extended stdlib builtin.

```
my $result = djb2 $x
# or in a pipeline:
@list |> map djb2 |> p
```

djb2_hash

djb2_hash — batch 11 builtin.

```
my $result = djb2_hash $x
# or in a pipeline:
@list |> map djb2_hash |> p
```

dm

dm — python/ruby stdlib builtin. Alias for divmod.

```
my $result = dm $x
# or in a pipeline:
@list |> map dm |> p
```

dmetph

dmetph — encoding / phonetics builtin. Alias for double_metaphone.

```
my $result = dmetph $x
# or in a pipeline:
@list |> map dmetph |> p
```

dnbinom

dnbinom — negative binomial PMF. Args: k, size, prob.

```
p dnbinom(3, 5, 0.5)  # P(3 failures before 5 successes)
```

dnds_ratio

dnds_ratio — batch 33 builtin.

```
my $result = dnds_ratio $x
# or in a pipeline:
@list |> map dnds_ratio |> p
```

dnorm

dnorm — batch 53 builtin.

```
my $result = dnorm $x
# or in a pipeline:
@list |> map dnorm |> p
```

dnrm2

dnrm2 — batch 66 builtin.

```
my $result = dnrm2 $x
# or in a pipeline:
@list |> map dnrm2 |> p
```

do_call

do_call — call a function with args from a list. Like R's do.call().

```
my $result = docall(fn { $_[0] + $_[1] }, [3, 4])  # 7
```

doctor

doctor (alias health) — runtime diagnostic report. Prints version + binary path, runtime flags (--compat, --no-interop, big-int, strict_vars, strict_refs), environment overrides (STRYKE_NO_TTY, STRYKE_NO_JIT, RUST_LOG, ...), reflection counts (%b primaries, %all spellings, categories, list-builtin extras), concurrency (logical CPUs + rayon pool size), home/cache/config dirs, toolchain availability (rustc/cargo/perl/git/rg), and a battery of sanity checks. Returns the warning count as an integer (0 = healthy).

```
doctor              # full report
my $w = doctor
exit($w == 0 ? 0 : 1)   # CI-friendly health gate
```

dodecahedral_number

dodecahedral_number — batch 11 builtin.

```
my $result = dodecahedral_number $x
# or in a pipeline:
@list |> map dodecahedral_number |> p
```

dodgson_swap_count

dodgson_swap_count — batch 43 builtin.

```
my $result = dodgson_swap_count $x
# or in a pipeline:
@list |> map dodgson_swap_count |> p
```

dogleg_step

dogleg_step — batch 19 builtin.

```
my $result = dogleg_step $x
# or in a pipeline:
@list |> map dogleg_step |> p
```

dollarde

dollarde — batch 81 builtin.

```
my $result = dollarde $x
# or in a pipeline:
@list |> map dollarde |> p
```

dollarfr

dollarfr — batch 81 builtin.

```
my $result = dollarfr $x
# or in a pipeline:
@list |> map dollarfr |> p
```

dom_tree_idom

dom_tree_idom — batch 68 builtin.

```
my $result = dom_tree_idom $x
# or in a pipeline:
@list |> map dom_tree_idom |> p
```

dominance_frontier

`dominance_frontier` — batch 68 builtin.

```
my $result = dominance_frontier $x
# or in a pipeline:
@list |> map dominance_frontier |> p
```

dominating_set_greedy

`dominating_set_greedy` — batch 75 builtin.

```
my $result = dominating_set_greedy $x
# or in a pipeline:
@list |> map dominating_set_greedy |> p
```

dominating_set_greedy_step

`dominating_set_greedy_step` — batch 41 builtin.

```
my $result = dominating_set_greedy_step $x
# or in a pipeline:
@list |> map dominating_set_greedy_step |> p
```

dominating_set_lp_bound

`dominating_set_lp_bound` — batch 41 builtin.

```
my $result = dominating_set_lp_bound $x
# or in a pipeline:
@list |> map dominating_set_lp_bound |> p
```

donaldson_invariant

`donaldson_invariant` — batch 37 builtin.

```
my $result = donaldson_invariant $x
# or in a pipeline:
@list |> map donaldson_invariant |> p
```

doppler

`doppler_frequency($source_freq, $source_vel, $observer_vel)` (alias `doppler`) — observed frequency with Doppler effect. Uses speed of sound (343 m/s).

```
p doppler(440, -30, 0)   # ~482 Hz (ambulance approaching at 30 m/s)
p doppler(440, 30, 0)    # ~405 Hz (ambulance receding)
```

doppler_classical

`doppler_classical` — batch 23 builtin.

```
my $result = doppler_classical $x
# or in a pipeline:
@list |> map doppler_classical |> p
```

doppler_relativistic

`doppler_relativistic` — batch 12 builtin.

```
my $result = doppler_relativistic $x
# or in a pipeline:
@list |> map doppler_relativistic |> p
```

dopri5_combine

`dopri5_combine` — batch 34 builtin.

```
my $result = dopri5_combine $x
# or in a pipeline:
@list |> map dopri5_combine |> p
```

dorglq

`dorglq` — batch 66 builtin.

```
my $result = dorglq $x
# or in a pipeline:
@list |> map dorglq |> p
```

dorgqr

`dorgqr` — batch 66 builtin.

```
my $result = dorgqr $x
# or in a pipeline:
@list |> map dorgqr |> p
```

dot_case

`dot_case` — string (batch 2) builtin.

```
my $result = dot_case $x
# or in a pipeline:
@list |> map dot_case |> p
```

dot_product

`dot_product` — extended stdlib builtin.

```
my $result = dot_product $x
# or in a pipeline:
@list |> map dot_product |> p
```

dotp

`dotp` — extended stdlib builtin. Alias for `dot_product`.

```
my $result = dotp $x
# or in a pipeline:
@list |> map dotp |> p
```

double

`double` — trivial numeric / predicate builtins builtin.

```
my $result = double $x
# or in a pipeline:
@list |> map double |> p
```

double_auction_step

`double_auction_step` — batch 43 builtin.

```
my $result = double_auction_step $x
# or in a pipeline:
@list |> map double_auction_step |> p
```

double_each

`double_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = double_each $x
# or in a pipeline:
@list |> map double_each |> p
```

double_factorial

`double_factorial` — math formulas builtin.

```
my $result = double_factorial $x
# or in a pipeline:
@list |> map double_factorial |> p
```

double_layer_capacitance

`double_layer_capacitance` — batch 38 builtin.

```
my $result = double_layer_capacitance $x
# or in a pipeline:
@list |> map double_layer_capacitance |> p
```

double_metaphone

`double_metaphone` — encoding / phonetics builtin.

```
my $result = double_metaphone $x
# or in a pipeline:
@list |> map double_metaphone |> p
```

double_q_learning_step

`double_q_learning_step` — batch 40 builtin.

```
my $result = double_q_learning_step $x
# or in a pipeline:
@list |> map double_q_learning_step |> p
```

doubling_time_growth

`doubling_time_growth` — batch 61 builtin.

```
my $result = doubling_time_growth $x
# or in a pipeline:
@list |> map doubling_time_growth |> p
```

downcase_each

downcase_each — trivial numeric helpers (batch 4) builtin.

```
my $result = downcase_each $x
# or in a pipeline:
@list |> map downcase_each |> p
```

downsample

downsample(\@signal, $factor) (alias decimate) — reduces sample rate by keeping every nth sample. Apply anti-aliasing filter first to avoid aliasing.

```
my $decimated = decimate(\@signal, 4)  # keep every 4th sample
```

downto

downto — python/ruby stdlib builtin.

```
my $result = downto $x
# or in a pipeline:
@list |> map downto |> p
```

doy

doy — extended stdlib batch 3 builtin. Alias for day_of_year.

```
my $result = doy $x
# or in a pipeline:
@list |> map doy |> p
```

dpayback

dpayback — finance (extended) builtin. Alias for discounted_payback.

```
my $result = dpayback $x
# or in a pipeline:
@list |> map dpayback |> p
```

dpbsv

dpbsv — batch 66 builtin.

```
my $result = dpbsv $x
# or in a pipeline:
@list |> map dpbsv |> p
```

dpll_clause_learning

dpll_clause_learning — batch 67 builtin.

```
my $result = dpll_clause_learning $x
# or in a pipeline:
@list |> map dpll_clause_learning |> p
```

dpotrf

dpotrf — batch 66 builtin.

```
my $result = dpotrf $x
# or in a pipeline:
@list |> map dpotrf |> p
```

dpss_window

`dpss_window` — batch 74 builtin.

```
my $result = dpss_window $x
# or in a pipeline:
@list |> map dpss_window |> p
```

dr

`dr` — filesystem extensions builtin.

```
my $result = dr $x
# or in a pipeline:
@list |> map dr |> p
```

drag_force

`drag_force` (alias `fdrag`) computes aerodynamic drag: F = 0.5·Cd·ρ·A·v². Args: drag_coeff, air_density, area, velocity.

```
p fdrag(0.47, 1.225, 0.01, 30)  # drag on a ball at 30 m/s
```

drag_force_quadratic

`drag_force_quadratic` — batch 12 builtin.

```
my $result = drag_force_quadratic $x
# or in a pipeline:
@list |> map drag_force_quadratic |> p
```

drnk

`drnk` — extended stdlib builtin. Alias for `dense_rank`.

```
my $result = drnk $x
# or in a pipeline:
@list |> map drnk |> p
```

drop_at

`drop_at` — batch 9 builtin.

```
my $result = drop_at $x
# or in a pipeline:
@list |> map drop_at |> p
```

drop_every

`drop_every` — list helpers (batch 4) builtin.

```
my $result = drop_every $x
# or in a pipeline:
@list |> map drop_every |> p
```

drop_last

`drop_last` — file stat / path builtin.

```
my $result = drop_last $x
# or in a pipeline:
@list |> map drop_last |> p
```

drop_n

`drop_n` — collection helpers (trivial) builtin.

```
my $result = drop_n $x
# or in a pipeline:
@list |> map drop_n |> p
```

drot

`drot` — batch 66 builtin.

```
my $result = drot $x
# or in a pipeline:
@list |> map drot |> p
```

drotg

`drotg` — batch 66 builtin.

```
my $result = drotg $x
# or in a pipeline:
@list |> map drotg |> p
```

dsamp

`dsamp` — dsp / signal (extended) builtin. Alias for `downsample`.

```
my $result = dsamp $x
# or in a pipeline:
@list |> map dsamp |> p
```

dscal

`dscal` — batch 66 builtin.

```
my $result = dscal $x
# or in a pipeline:
@list |> map dscal |> p
```

dsyevd

`dsyevd` — batch 66 builtin.

```
my $result = dsyevd $x
# or in a pipeline:
@list |> map dsyevd |> p
```

dsyrk

`dsyrk` — batch 66 builtin.

```
my $result = dsyrk $x
# or in a pipeline:
@list |> map dsyrk |> p
```

dt

`dt` — batch 53 builtin.

```
my $result = dt $x
# or in a pipeline:
@list |> map dt |> p
```

dtbsv

`dtbsv` — batch 66 builtin.

```
my $result = dtbsv $x
# or in a pipeline:
@list |> map dtbsv |> p
```

dte

`dte` — date / time builtin. Alias for `datetime_from_epoch`.

```
my $result = dte $x
# or in a pipeline:
@list |> map dte |> p
```

dtf

`dtf` — date / time builtin. Alias for `datetime_strftime`.

```
my $result = dtf $x
# or in a pipeline:
@list |> map dtf |> p
```

dtrsm

`dtrsm` — batch 66 builtin.

```
my $result = dtrsm $x
# or in a pipeline:
@list |> map dtrsm |> p
```

dtrsv

`dtrsv` — batch 66 builtin.

```
my $result = dtrsv $x
# or in a pipeline:
@list |> map dtrsv |> p
```

dump

`dump` — process / system builtin.

```
my $result = dump $x
# or in a pipeline:
@list |> map dump |> p
```

dunif

`dunif` — uniform PDF. Args: x, a, b.

```
p dunif(0.5, 0, 1)  # 1.0
```

duped

`duplicated` — boolean array: 1 if element appeared earlier in the vector, 0 otherwise. Like R's duplicated().

```
p duplicated([1,2,3,2,1])  # [0,0,0,1,1]
```

duplicate_count

`duplicate_count` — list helpers (batch 4) builtin.

```
my $result = duplicate_count $x
# or in a pipeline:
@list |> map duplicate_count |> p
```

durand_kerner

`durand_kerner` — batch 10 builtin. Alias for `polynomial_roots_dk`.

```
my $result = durand_kerner $x
# or in a pipeline:
@list |> map durand_kerner |> p
```

duration_macaulay

`duration_macaulay` — batch 81 builtin.

```
my $result = duration_macaulay $x
# or in a pipeline:
@list |> map duration_macaulay |> p
```

durbin_watson_d

`durbin_watson_d` — batch 78 builtin.

```
my $result = durbin_watson_d $x
# or in a pipeline:
@list |> map durbin_watson_d |> p
```

dutch_auction

`dutch_auction` — batch 73 builtin.

```
my $result = dutch_auction $x
# or in a pipeline:
@list |> map dutch_auction |> p
```

dutch_auction_step

`dutch_auction_step` — batch 43 builtin.

```
my $result = dutch_auction_step $x
# or in a pipeline:
@list |> map dutch_auction_step |> p
```

dynamics_db_level

`dynamics_db_level` — batch 64 builtin.

```
my $result = dynamics_db_level $x
# or in a pipeline:
@list |> map dynamics_db_level |> p
```

each_cons

`each_cons` — python/ruby stdlib builtin.

```
my $result = each_cons $x
# or in a pipeline:
@list |> map each_cons |> p
```

each_slice

`each_slice` — python/ruby stdlib builtin.

```
my $result = each_slice $x
# or in a pipeline:
@list |> map each_slice |> p
```

each_with_index

`each_with_index` — functional / iterator builtin.

```
my $result = each_with_index $x
# or in a pipeline:
@list |> map each_with_index |> p
```

each_with_object

`each_with_object` — additional missing stdlib functions builtin.

```
my $result = each_with_object $x
# or in a pipeline:
@list |> map each_with_object |> p
```

eadie_hofstee_y

`eadie_hofstee_y` — batch 12 builtin.

```
my $result = eadie_hofstee_y $x
# or in a pipeline:
@list |> map eadie_hofstee_y |> p
```

ean13_check

`ean13_check` — batch 82 builtin.

```
my $result = ean13_check $x
# or in a pipeline:
@list |> map ean13_check |> p
```

earley_complete

`earley_complete` — batch 68 builtin.

```
my $result = earley_complete $x
# or in a pipeline:
@list |> map earley_complete |> p
```

earley_predict

`earley_predict` — batch 68 builtin.

```
my $result = earley_predict $x
# or in a pipeline:
@list |> map earley_predict |> p
```

earley_scan

`earley_scan` — batch 68 builtin.

```
my $result = earley_scan $x
# or in a pipeline:
@list |> map earley_scan |> p
```

earth_mass

`earth_mass` (alias `mearth`) — M🜨 5.972×10²⁴ kg. Mass of Earth.

```
p mearth   # 5.972167867e24 kg
```

earth_radius

`earth_radius` — physics constants builtin.

```
my $result = earth_radius $x
# or in a pipeline:
@list |> map earth_radius |> p
```

easter_gregorian_year

`easter_gregorian_year` — batch 52 builtin.

```
my $result = easter_gregorian_year $x
# or in a pipeline:
@list |> map easter_gregorian_year |> p
```

easter_julian_year

`easter_julian_year` — batch 52 builtin.

```
my $result = easter_julian_year $x
# or in a pipeline:
@list |> map easter_julian_year |> p
```

easter_orthodox_year

`easter_orthodox_year` — batch 52 builtin.

```
my $result = easter_orthodox_year $x
# or in a pipeline:
@list |> map easter_orthodox_year |> p
```

easter_orthodox_year_2

`easter_orthodox_year_2` — batch 82 builtin.

```
my $result = easter_orthodox_year_2 $x
# or in a pipeline:
@list |> map easter_orthodox_year_2 |> p
```

easter_sunday

`easter_sunday` — batch 9 builtin.

```
my $result = easter_sunday $x
# or in a pipeline:
@list |> map easter_sunday |> p
```

easter_western

`easter_western` — batch 82 builtin.

```
my $result = easter_western $x
# or in a pipeline:
@list |> map easter_western |> p
```

ecc_point_double

`ecc_point_double` — batch 35 builtin.

```
my $result = ecc_point_double $x
# or in a pipeline:
@list |> map ecc_point_double |> p
```

ecdf

`ecdf` — empirical CDF: proportion of data ≤ x. Like R's ecdf().

```
p ecdf([1,2,3,4,5], 3)  # 0.6 (3 of 5 values ≤ 3)
```

echarge

`echarge` — constants builtin. Alias for `elementary_charge`.

```
my $result = echarge $x
# or in a pipeline:
@list |> map echarge |> p
```

eclipse_magnitude

`eclipse_magnitude` — batch 55 builtin.

```
my $result = eclipse_magnitude $x
# or in a pipeline:
@list |> map eclipse_magnitude |> p
```

ecliptic_obliquity

`ecliptic_obliquity` — batch 9 builtin.

```
my $result = ecliptic_obliquity $x
# or in a pipeline:
@list |> map ecliptic_obliquity |> p
```

ecliptic_to_equatorial

`ecliptic_to_equatorial` — batches 14-16 builtin.

```
my $result = ecliptic_to_equatorial $input
```

econs

`econs` — python/ruby stdlib builtin. Alias for `each_cons`.

```
my $result = econs $x
# or in a pipeline:
@list |> map econs |> p
```

eddington_luminosity

`eddington_luminosity` — batch 12 builtin.

```
my $result = eddington_luminosity $x
# or in a pipeline:
@list |> map eddington_luminosity |> p
```

edge_coloring_vizing_step

`edge_coloring_vizing_step` — batch 41 builtin.

```
my $result = edge_coloring_vizing_step $x
# or in a pipeline:
@list |> map edge_coloring_vizing_step |> p
```

edge_connectivity

`edge_connectivity` — batch 75 builtin.

```
my $result = edge_connectivity $x
# or in a pipeline:
@list |> map edge_connectivity |> p
```

edge_detect_kernel

`edge_detect_kernel` — batches 14-16 builtin.

```
my $result = edge_detect_kernel $x
# or in a pipeline:
@list |> map edge_detect_kernel |> p
```

edgeworth_box_alloc

`edgeworth_box_alloc` — batch 73 builtin.

```
my $result = edgeworth_box_alloc $x
# or in a pipeline:
@list |> map edgeworth_box_alloc |> p
```

edmonds_karp_bfs

`edmonds_karp_bfs` — batch 75 builtin.

```
my $result = edmonds_karp_bfs $x
# or in a pipeline:
@list |> map edmonds_karp_bfs |> p
```

edmonds_karp_step

`edmonds_karp_step` — batch 41 builtin.

```
my $result = edmonds_karp_step $x
# or in a pipeline:
@list |> map edmonds_karp_step |> p
```

effect

`effect` — batch 81 builtin.

```
my $result = effect $x
# or in a pipeline:
@list |> map effect |> p
```

effective_pop_size

`effective_pop_size` — batch 22 builtin.

```
my $result = effective_pop_size $x
# or in a pipeline:
@list |> map effective_pop_size |> p
```

efficiency_ratio

`efficiency_ratio` — batch 12 builtin.

```
my $result = efficiency_ratio $x
# or in a pipeline:
@list |> map efficiency_ratio |> p
```

efield

`efield` — physics formulas builtin. Alias for `electric_field`.

```
my $result = efield $x
# or in a pipeline:
@list |> map efield |> p
```

efield_point

`efield_point` — batch 23 builtin.

```
my $result = efield_point $x
# or in a pipeline:
@list |> map efield_point |> p
```

egalitarian_solution

`egalitarian_solution` — batch 43 builtin.

```
my $result = egalitarian_solution $x
# or in a pipeline:
@list |> map egalitarian_solution |> p
```

egalitarian_split

`egalitarian_split` — batch 43 builtin.

```
my $result = egalitarian_split $x
# or in a pipeline:
@list |> map egalitarian_split |> p
```

egarch_step

`egarch_step` — batches 17-18 builtin.

```
my $result = egarch_step $x
# or in a pipeline:
@list |> map egarch_step |> p
```

eig

`matrix_eigenvalues` (aliases `meig`, `eig`) computes eigenvalues of a square matrix via QR iteration. Returns an array of eigenvalues.

```
my @eigs = @{eig([[2,1],[1,2]])}   # [3, 1]
```

einstein_dilaton_step

einstein_dilaton_step — batch 39 builtin.

```
my $result = einstein_dilaton_step $x
# or in a pipeline:
@list |> map einstein_dilaton_step |> p
```

einstein_ring_radius

einstein_ring_radius — batch 31 builtin.

```
my $result = einstein_ring_radius $x
# or in a pipeline:
@list |> map einstein_ring_radius |> p
```

einstein_tensor_step

einstein_tensor_step — batch 39 builtin.

```
my $result = einstein_tensor_step $x
# or in a pipeline:
@list |> map einstein_tensor_step |> p
```

either

either — boolean combinators builtin.

```
my $result = either $x
# or in a pipeline:
@list |> map either |> p
```

ekf_jacobian_step

ekf_jacobian_step — batch 40 builtin.

```
my $result = ekf_jacobian_step $x
# or in a pipeline:
@list |> map ekf_jacobian_step |> p
```

ekman_layer_depth

ekman_layer_depth — batch 42 builtin.

```
my $result = ekman_layer_depth $x
# or in a pipeline:
@list |> map ekman_layer_depth |> p
```

ekman_pumping_step

ekman_pumping_step — batch 42 builtin.

```
my $result = ekman_pumping_step $x
# or in a pipeline:
@list |> map ekman_pumping_step |> p
```

electric_field

electric_field — physics formulas builtin.

```
my $result = electric_field $x
# or in a pipeline:
@list |> map electric_field |> p
```

electrochem_impedance_z

`electrochem_impedance_z` — batch 38 builtin.

```
my $result = electrochem_impedance_z $x
# or in a pipeline:
@list |> map electrochem_impedance_z |> p
```

electrochemiluminescence_yield

`electrochemiluminescence_yield` — batch 38 builtin.

```
my $result = electrochemiluminescence_yield $x
# or in a pipeline:
@list |> map electrochemiluminescence_yield |> p
```

electrocrystallization_step

`electrocrystallization_step` — batch 38 builtin.

```
my $result = electrocrystallization_step $x
# or in a pipeline:
@list |> map electrocrystallization_step |> p
```

electrode_potential_step

`electrode_potential_step` — batch 38 builtin.

```
my $result = electrode_potential_step $x
# or in a pipeline:
@list |> map electrode_potential_step |> p
```

electrolysis_mass

`electrolysis_mass` — batch 63 builtin.

```
my $result = electrolysis_mass $x
# or in a pipeline:
@list |> map electrolysis_mass |> p
```

electrolyte_decomposition_temp

`electrolyte_decomposition_temp` — batch 38 builtin.

```
my $result = electrolyte_decomposition_temp $x
# or in a pipeline:
@list |> map electrolyte_decomposition_temp |> p
```

electrolyzer_voltage

`electrolyzer_voltage` — batch 38 builtin.

```
my $result = electrolyzer_voltage $x
# or in a pipeline:
@list |> map electrolyzer_voltage |> p
```

electron_mass

`electron_mass` — constants builtin.

```
my $result = electron_mass $x
# or in a pipeline:
@list |> map electron_mass |> p
```

electroosmotic_velocity

`electroosmotic_velocity` — batch 38 builtin.

```
my $result = electroosmotic_velocity $x
# or in a pipeline:
@list |> map electroosmotic_velocity |> p
```

elem_at

`elem_at` — list helpers (batch 4) builtin.

```
my $result = elem_at $x
# or in a pipeline:
@list |> map elem_at |> p
```

elem_index

`elem_index` — additional missing stdlib functions builtin.

```
my $result = elem_index $x
# or in a pipeline:
@list |> map elem_index |> p
```

elem_indices

`elem_indices` — additional missing stdlib functions builtin.

```
my $result = elem_indices $x
# or in a pipeline:
@list |> map elem_indices |> p
```

elem_of

`elem_of` — haskell list functions builtin.

```
my $result = elem_of $x
# or in a pipeline:
@list |> map elem_of |> p
```

elementary_charge

`elementary_charge` — constants builtin.

```
my $result = elementary_charge $x
# or in a pipeline:
@list |> map elementary_charge |> p
```

elgamal_encrypt

`elgamal_encrypt` — batch 35 builtin.

```
my $result = elgamal_encrypt $x
# or in a pipeline:
@list |> map elgamal_encrypt |> p
```

elias_delta_code

`elias_delta_code` — batch 40 builtin.

```
my $result = elias_delta_code $x
# or in a pipeline:
@list |> map elias_delta_code |> p
```

elias_gamma_code

`elias_gamma_code` — batch 40 builtin.

```
my $result = elias_gamma_code $x
# or in a pipeline:
@list |> map elias_gamma_code |> p
```

elidx

`elidx` — additional missing stdlib functions builtin. Alias for `elem_index`.

```
my $result = elidx $x
# or in a pipeline:
@list |> map elidx |> p
```

elidxs

`elidxs` — additional missing stdlib functions builtin. Alias for `elem_indices`.

```
my $result = elidxs $x
# or in a pipeline:
@list |> map elidxs |> p
```

ellip_lp

`ellip_lp` — batch 74 builtin.

```
my $result = ellip_lp $x
# or in a pipeline:
@list |> map ellip_lp |> p
```

ellipe

`ellipe` — batch 72 builtin.

```
my $result = ellipe $x
# or in a pipeline:
@list |> map ellipe |> p
```

ellipk

`ellipk` — batch 72 builtin.

```
my $result = ellipk $x
# or in a pipeline:
@list |> map ellipk |> p
```

ellipperim

`ellipperim` — geometry (extended) builtin. Alias for `ellipse_perimeter`.

```
my $result = ellipperim $x
# or in a pipeline:
@list |> map ellipperim |> p
```

ellipse_perimeter

`ellipse_perimeter($a, $b)` (alias `ellper`) — approximates the perimeter of an ellipse with semi-axes a and b. Uses Ramanujan's approximation.

```
p ellper(5, 3)   # ~25.5
p ellper(10, 10)   # ~62.8 (circle)
```

ellipsis

`ellipsis` — string quote / escape builtin.

```
my $result = ellipsis $x
# or in a pipeline:
@list |> map ellipsis |> p
```

ellipsoid_surface_approx

`ellipsoid_surface_approx` — batch 28 builtin.

```
my $result = ellipsoid_surface_approx $x
# or in a pipeline:
@list |> map ellipsoid_surface_approx |> p
```

ellipsoid_volume

`ellipsoid_volume` — batch 28 builtin.

```
my $result = ellipsoid_volume $x
# or in a pipeline:
@list |> map ellipsoid_volume |> p
```

elo_expected

`elo_expected` — batch 56 builtin.

```
my $result = elo_expected $x
# or in a pipeline:
@list |> map elo_expected |> p
```

elo_update

`elo_update` — batch 56 builtin.

```
my $result = elo_update $x
# or in a pipeline:
@list |> map elo_update |> p
```

elof

`elof` — haskell list functions builtin. Alias for `elem_of`.

```
my $result = elof $x
# or in a pipeline:
@list |> map elof |> p
```

elu

`elu` applies the Exponential Linear Unit: x if x□0, alpha*(e^x - 1) otherwise.

```
p elu(1)     # 1
p elu(-1)    # -0.632
```

em_frequency

`em_frequency` — batch 23 builtin.

```
my $result = em_frequency $x
# or in a pipeline:
@list |> map em_frequency |> p
```

em_intensity

`em_intensity` — batch 23 builtin.

```
my $result = em_intensity $x
# or in a pipeline:
@list |> map em_intensity |> p
```

em_step

`em_step` — batch 34 builtin.

```
my $result = em_step $x
# or in a pipeline:
@list |> map em_step |> p
```

em_wavelength

`em_wavelength` — batch 23 builtin.

```
my $result = em_wavelength $x
# or in a pipeline:
@list |> map em_wavelength |> p
```

email_domain

`email_domain` — url / email parts builtin.

```
my $result = email_domain $x
# or in a pipeline:
@list |> map email_domain |> p
```

email_local

`email_local` — url / email parts builtin.

```
my $result = email_local $x
# or in a pipeline:
@list |> map email_local |> p
```

emass

`emass` — constants builtin. Alias for `electron_mass`.

```
my $result = emass $x
# or in a pipeline:
@list |> map emass |> p
```

emavg

`emavg` — math / numeric (uncategorized batch) builtin. Alias for `exponential_moving_average`.

```
my $result = emavg $x
# or in a pipeline:
@list |> map emavg |> p
```

embed_ts

`embed_ts` — time-delay embedding. Converts a time series into a matrix of lagged values. Like R's embed().

```
p embed_ts([1,2,3,4,5], 3)  # [[3,2,1],[4,3,2],[5,4,3]]
```

emboss_kernel

`emboss_kernel` — batches 14-16 builtin.

```
my $result = emboss_kernel $x
# or in a pipeline:
@list |> map emboss_kernel |> p
```

emf_from_half_cells

`emf_from_half_cells` — batch 21 builtin.

```
my $result = emf_from_half_cells $x
# or in a pipeline:
@list |> map emf_from_half_cells |> p
```

emissivity_grey_body

`emissivity_grey_body` — batch 42 builtin.

```
my $result = emissivity_grey_body $x
# or in a pipeline:
@list |> map emissivity_grey_body |> p
```

emitted_long_wave

`emitted_long_wave` — batch 42 builtin.

```
my $result = emitted_long_wave $x
# or in a pipeline:
@list |> map emitted_long_wave |> p
```

emojinum

`emojinum` — encoding / phonetics builtin. Alias for `to_emoji_num`.

```
my $result = emojinum $x
# or in a pipeline:
@list |> map emojinum |> p
```

empc

`empc` — algebraic match builtin. Alias for `empty_clj`.

```
my $result = empc $x
# or in a pipeline:
@list |> map empc |> p
```

empty_clj

`empty_clj` — algebraic match builtin.

```
my $result = empty_clj $x
# or in a pipeline:
@list |> map empty_clj |> p
```

empty_count

`empty_count` — counters (batch 4) builtin.

```
my $result = empty_count $x
# or in a pipeline:
@list |> map empty_count |> p
```

encode_base

`encode_base` — batch 54 builtin.

```
my $result = encode_base $x
# or in a pipeline:
@list |> map encode_base |> p
```

encode_pair

`encode_pair` — batch 67 builtin.

```
my $result = encode_pair $x
# or in a pipeline:
@list |> map encode_pair |> p
```

encode_succ

`encode_succ` — batch 67 builtin.

```
my $result = encode_succ $x
# or in a pipeline:
@list |> map encode_succ |> p
```

end_of_day

`end_of_day` — extended stdlib builtin.

```
my $result = end_of_day $x
# or in a pipeline:
@list |> map end_of_day |> p
```

endgrent

`endgrent` — posix metadata builtin.

```
my $result = endgrent $x
# or in a pipeline:
@list |> map endgrent |> p
```

endhostent

`endhostent` — posix metadata builtin.

```
my $result = endhostent $x
# or in a pipeline:
@list |> map endhostent |> p
```

endianness

`endianness` — byte order of the platform: `"little"` or `"big"`. Compile-time constant.

```
p endianness                            # little
```

endnetent

`endnetent` — posix metadata builtin.

```
my $result = endnetent $x
# or in a pipeline:
@list |> map endnetent |> p
```

endowment_combined_a

`endowment_combined_a` — batch 60 builtin.

```
my $result = endowment_combined_a $x
# or in a pipeline:
@list |> map endowment_combined_a |> p
```

endowment_pure_e

`endowment_pure_e` — batch 60 builtin.

```
my $result = endowment_pure_e $x
# or in a pipeline:
@list |> map endowment_pure_e |> p
```

endprotoent

`endprotoent` — posix metadata builtin.

```
my $result = endprotoent $x
# or in a pipeline:
@list |> map endprotoent |> p
```

endpwent

`endpwent` — posix metadata builtin.

```
my $result = endpwent $x
# or in a pipeline:
@list |> map endpwent |> p
```

ends_with

`ends_with` — trivial string ops builtin.

```
my $result = ends_with $x
# or in a pipeline:
@list |> map ends_with |> p
```

ends_with_any

`ends_with_any` — string (batch 2) builtin.

```
my $result = ends_with_any $x
# or in a pipeline:
@list |> map ends_with_any |> p
```

endservent

`endservent` — posix metadata builtin.

```
my $result = endservent $x
# or in a pipeline:
@list |> map endservent |> p
```

energy

`energy(\@signal)` — computes the total energy of a signal as the sum of squared samples. Useful for audio analysis and feature extraction.

```
p energy(\@signal)   # total signal energy
```

energy_density_battery

`energy_density_battery` — batch 38 builtin.

```
my $result = energy_density_battery $x
# or in a pipeline:
@list |> map energy_density_battery |> p
```

energy_efficiency_cell

`energy_efficiency_cell` — batch 38 builtin.

```
my $result = energy_efficiency_cell $x
# or in a pipeline:
@list |> map energy_efficiency_cell |> p
```

engel_curve

`engel_curve` — batch 73 builtin.

```
my $result = engel_curve $x
# or in a pipeline:
@list |> map engel_curve |> p
```

engle_granger_step

`engle_granger_step` — batch 36 builtin.

```
my $result = engle_granger_step $x
# or in a pipeline:
@list |> map engle_granger_step |> p
```

english_auction

`english_auction` — batch 73 builtin.

```
my $result = english_auction $x
# or in a pipeline:
@list |> map english_auction |> p
```

english_auction_step

`english_auction_step` — batch 43 builtin.

```
my $result = english_auction_step $x
# or in a pipeline:
@list |> map english_auction_step |> p
```

english_chi2

`english_chi2` — batch 26 builtin.

```
my $result = english_chi2 $x
# or in a pipeline:
@list |> map english_chi2 |> p
```

english_likeness

`english_likeness` — batch 35 builtin.

```
my $result = english_likeness $x
# or in a pipeline:
@list |> map english_likeness |> p
```

enso_index_step

`enso_index_step` — batch 42 builtin.

```
my $result = enso_index_step $x
# or in a pipeline:
@list |> map enso_index_step |> p
```

ensure_prefix

`ensure_prefix` — path helpers builtin.

```
my $result = ensure_prefix $x
# or in a pipeline:
@list |> map ensure_prefix |> p
```

ensure_suffix

`ensure_suffix` — path helpers builtin.

```
my $result = ensure_suffix $x
# or in a pipeline:
@list |> map ensure_suffix |> p
```

entanglement_entropy

`entanglement_entropy` — batch 80 builtin.

```
my $result = entanglement_entropy $x
# or in a pipeline:
@list |> map entanglement_entropy |> p
```

enthalpy_reaction

`enthalpy_reaction` — batch 21 builtin.

```
my $result = enthalpy_reaction $x
# or in a pipeline:
@list |> map enthalpy_reaction |> p
```

entropy

`entropy` — math functions builtin.

```
my $result = entropy $x
# or in a pipeline:
@list |> map entropy |> p
```

entropy_bits

`entropy_bits` — batch 27 builtin.

```
my $result = entropy_bits $x
# or in a pipeline:
@list |> map entropy_bits |> p
```

env

`env` — dsp / signal (extended) builtin. Alias for `envelope`.

```
my $result = env $x
# or in a pipeline:
@list |> map env |> p
```

env_get

`env_get` — process / env builtin.

```
my $result = env_get $x
# or in a pipeline:
@list |> map env_get |> p
```

env_has

`env_has` — process / env builtin.

```
my $result = env_has $x
# or in a pipeline:
@list |> map env_has |> p
```

env_keys

`env_keys` — process / env builtin.

```
my $result = env_keys $x
# or in a pipeline:
@list |> map env_keys |> p
```

env_pairs

`env_pairs` — more process / system builtin.

```
my $result = env_pairs $x
# or in a pipeline:
@list |> map env_pairs |> p
```

env_remove

`env_remove` — more process / system builtin.

```
my $result = env_remove $x
# or in a pipeline:
@list |> map env_remove |> p
```

env_set

`env_set` — more process / system builtin.

```
my $result = env_set $x
# or in a pipeline:
@list |> map env_set |> p
```

envelope

`envelope(\@signal)` (alias `hilbert_env`) — computes the amplitude envelope using Hilbert transform magnitude. Useful for amplitude modulation analysis.

```
my $env = envelope(\@signal)
# $env traces the amplitude of oscillations
```

envelope_amplitude

`envelope_amplitude` — batch 74 builtin.

```
my $result = envelope_amplitude $x
# or in a pipeline:
@list |> map envelope_amplitude |> p
```

eod

`eod` — extended stdlib builtin. Alias for `end_of_day`.

```
my $result = eod $x
# or in a pipeline:
@list |> map eod |> p
```

eof_from_concurrence

`eof_from_concurrence` — batch 32 builtin.

```
my $result = eof_from_concurrence $x
# or in a pipeline:
@list |> map eof_from_concurrence |> p
```

epa_per_play

`epa_per_play` — batch 56 builtin.

```
my $result = epa_per_play $x
# or in a pipeline:
@list |> map epa_per_play |> p
```

epoch

`epoch` — now / timestamp builtin. Alias for `unix_epoch`.

```
my $result = epoch $x
# or in a pipeline:
@list |> map epoch |> p
```

epoch_ms

`epoch_ms` — now / timestamp builtin. Alias for `unix_epoch_ms`.

```
my $result = epoch_ms $x
# or in a pipeline:
@list |> map epoch_ms |> p
```

epotential_point

`epotential_point` — batch 23 builtin.

```
my $result = epotential_point $x
# or in a pipeline:
@list |> map epotential_point |> p
```

eps

`eps` — math / numeric (uncategorized batch) builtin. Alias for `epsilon`.

```
my $result = eps $x
# or in a pipeline:
@list |> map eps |> p
```

epsilon

`epsilon` — math / numeric (uncategorized batch) builtin.

```
my $result = epsilon $x
# or in a pipeline:
@list |> map epsilon |> p
```

epsilon0

`vacuum_permittivity` (alias `epsilon0`) — ε_0 ≈ 8.854×10^{-12} F/m. Electric constant, permittivity of free space.

```
p epsilon0   # 8.8541878128e-12
```

epsilon_extinction

`epsilon_extinction` — batch 63 builtin.

```
my $result = epsilon_extinction $x
# or in a pipeline:
@list |> map epsilon_extinction |> p
```

eqidx

`eqidx` — extended stdlib batch 3 builtin. Alias for `equilibrium_index`.

```
my $result = eqidx $x
# or in a pipeline:
@list |> map eqidx |> p
```

eqrng

`eqrng` — extended stdlib builtin. Alias for `equal_range`.

```
my $result = eqrng $x
# or in a pipeline:
@list |> map eqrng |> p
```

equal_range

`equal_range` — extended stdlib builtin.

```
my $result = equal_range $x
# or in a pipeline:
@list |> map equal_range |> p
```

equal_tempered_freq

`equal_tempered_freq` — batch 64 builtin.

```
my $result = equal_tempered_freq $x
# or in a pipeline:
@list |> map equal_tempered_freq |> p
```

equation_of_time

`equation_of_time` — batch 9 builtin.

```
my $result = equation_of_time $x
# or in a pipeline:
@list |> map equation_of_time |> p
```

equatorial_to_galactic

`equatorial_to_galactic` — batches 14-16 builtin.

```
my $result = equatorial_to_galactic $input
```

equilibrium_index

`equilibrium_index` — extended stdlib batch 3 builtin.

```
my $result = equilibrium_index $x
# or in a pipeline:
@list |> map equilibrium_index |> p
```

equinox_spring

`equinox_spring` — batch 82 builtin.

```
my $result = equinox_spring $x
# or in a pipeline:
@list |> map equinox_spring |> p
```

equirectangular_project

`equirectangular_project` — batches 14-16 builtin.

```
my $result = equirectangular_project $x
# or in a pipeline:
@list |> map equirectangular_project |> p
```

equivalent_potential_temp

`equivalent_potential_temp` — batch 42 builtin.

```
my $result = equivalent_potential_temp $x
# or in a pipeline:
@list |> map equivalent_potential_temp |> p
```

era_plus

`era_plus` — batch 56 builtin.

```
my $result = era_plus $x
# or in a pipeline:
@list |> map era_plus |> p
```

erf_approx

`erf_approx($x)` — error function. Used in probability, statistics, and partial differential equations.

```
p erf(1)    # ~0.843
p erf(2)    # ~0.995
```

erfcx

`erfcx` — batch 72 builtin.

```
my $result = erfcx $x
# or in a pipeline:
@list |> map erfcx |> p
```

erfi

`erfi` — batch 72 builtin.

```
my $result = erfi $x
# or in a pipeline:
@list |> map erfi |> p
```

error_correction_step

`error_correction_step` — batch 36 builtin.

```
my $result = error_correction_step $x
# or in a pipeline:
@list |> map error_correction_step |> p
```

ertel_pv_step

`ertel_pv_step` — batch 42 builtin.

```
my $result = ertel_pv_step $x
# or in a pipeline:
@list |> map ertel_pv_step |> p
```

escape_json

`escape_json` — json helpers builtin.

```
my $result = escape_json $x
# or in a pipeline:
@list |> map escape_json |> p
```

escape_velocity

`escape_velocity($mass, $radius)` (alias `escvel`) — minimum velocity to escape gravitational field: v = √(2GM/r). Defaults to Earth values.

```
p escvel                        # ~11186 m/s (Earth)
p escvel(1.989e30, 6.96e8)      # ~617 km/s (Sun)
```

escape_velocity_body

`escape_velocity_body` — batch 55 builtin.

```
my $result = escape_velocity_body $x
# or in a pipeline:
@list |> map escape_velocity_body |> p
```

eslice

`eslice` — python/ruby stdlib builtin. Alias for `each_slice`.

```
my $result = eslice $x
# or in a pipeline:
@list |> map eslice |> p
```

et_freq_ratio

`et_freq_ratio` — batch 30 builtin.

```
my $result = et_freq_ratio $x
# or in a pipeline:
@list |> map et_freq_ratio |> p
```

etag_validate

`etag_validate` — batch 81 builtin.

```
my $result = etag_validate $x
# or in a pipeline:
@list |> map etag_validate |> p
```

etd_rk2

`etd_rk2` — batch 34 builtin.

```
my $result = etd_rk2 $x
# or in a pipeline:
@list |> map etd_rk2 |> p
```

eth_addr_check

`eth_addr_check` — batch 82 builtin.

```
my $result = eth_addr_check $x
# or in a pipeline:
@list |> map eth_addr_check |> p
```

ethiopic_from_fixed

`ethiopic_from_fixed` — batch 52 builtin.

```
my $result = ethiopic_from_fixed $x
# or in a pipeline:
@list |> map ethiopic_from_fixed |> p
```

etot

`etot` — extended stdlib batch 3 builtin. Alias for `euler_totient`.

```
my $result = etot $x
# or in a pipeline:
@list |> map etot |> p
```

eucdist

`eucdist` — math / numeric (uncategorized batch) builtin. Alias for `euclidean_distance`.

```
my $result = eucdist $x
# or in a pipeline:
@list |> map eucdist |> p
```

euclidean_distance

`euclidean_distance` — math / numeric (uncategorized batch) builtin.

```
my $result = euclidean_distance $x
# or in a pipeline:
@list |> map euclidean_distance |> p
```

euclidean_distance_nd

`euclidean_distance_nd` — batch 28 builtin.

```
my $result = euclidean_distance_nd $x
# or in a pipeline:
@list |> map euclidean_distance_nd |> p
```

euler_char_complex

`euler_char_complex` — batch 37 builtin.

```
my $result = euler_char_complex $x
# or in a pipeline:
@list |> map euler_char_complex |> p
```

euler_characteristic

`euler_characteristic` — batch 28 builtin.

```
my $result = euler_characteristic $x
# or in a pipeline:
@list |> map euler_characteristic |> p
```

euler_e

`euler_e` — math / numeric (uncategorized batch) builtin. Alias for `euler_number`.

```
my $result = euler_e $x
# or in a pipeline:
@list |> map euler_e |> p
```

euler_lte

`euler_lte` — batch 34 builtin.

```
my $result = euler_lte $x
# or in a pipeline:
@list |> map euler_lte |> p
```

euler_mascheroni

`euler_mascheroni` (alias `gamma_const`) — Euler-Mascheroni constant γ ≈ 0.5772. Appears in number theory and analysis.

```
p euler_mascheroni   # 0.5772156649015329
```

euler_method

`euler_ode` (alias `euler_method`) solves an ODE dy/dt = f(t,y) using the Euler method. Returns [[t,y], ...].

```
my $sol = euler_ode(fn { $_[1] }, 0, 1, 0.01, 100)  # exponential growth
```

euler_number

`euler_number` — math / numeric (uncategorized batch) builtin.

```
my $result = euler_number $x
# or in a pipeline:
@list |> map euler_number |> p
```

euler_totient

`euler_totient` — extended stdlib batch 3 builtin.

```
my $result = euler_totient $x
# or in a pipeline:
@list |> map euler_totient |> p
```

eulerian_path_q

`eulerian_path_q` — batch 13 builtin.

```
my $result = eulerian_path_q $x
# or in a pipeline:
@list |> map eulerian_path_q |> p
```

ev_to_joules

`ev_to_joules` — batch 30 builtin.

```
my $result = ev_to_joules $input
```

eval_rpn

`eval_rpn` — algorithms / puzzles builtin.

```
my $result = eval_rpn $x
# or in a pipeline:
@list |> map eval_rpn |> p
```

even

`even` — trivial numeric / predicate builtins builtin.

```
my $result = even $x
# or in a pipeline:
@list |> map even |> p
```

event_horizon_check

`event_horizon_check` — batch 39 builtin.

```
my $result = event_horizon_check $x
# or in a pipeline:
@list |> map event_horizon_check |> p
```

every_nth

`every_nth` — collection helpers (trivial) builtin.

```
my $result = every_nth $x
# or in a pipeline:
@list |> map every_nth |> p
```

evolutionary_stable_strategy

`evolutionary_stable_strategy` — batch 43 builtin.

```
my $result = evolutionary_stable_strategy $x
# or in a pipeline:
@list |> map evolutionary_stable_strategy |> p
```

ew

`ew` — trivial string ops builtin. Alias for `ends_with`.

```
my $result = ew $x
# or in a pipeline:
@list |> map ew |> p
```

ewma_smooth

`ewma_smooth` — batch 78 builtin.

```
my $result = ewma_smooth $x
# or in a pipeline:
@list |> map ewma_smooth |> p
```

ewo

`ewo` — additional missing stdlib functions builtin. Alias for `each_with_object`.

```
my $result = ewo $x
# or in a pipeline:
@list |> map ewo |> p
```

ex_ante_value_check

`ex_ante_value_check` — batch 43 builtin.

```
my $result = ex_ante_value_check $x
# or in a pipeline:
@list |> map ex_ante_value_check |> p
```

ex_post_value_check

`ex_post_value_check` — batch 43 builtin.

```
my $result = ex_post_value_check $x
# or in a pipeline:
@list |> map ex_post_value_check |> p
```

exchange_current_density

`exchange_current_density` — batch 38 builtin.

```
my $result = exchange_current_density $x
# or in a pipeline:
@list |> map exchange_current_density |> p
```

exe

`exe` — filesystem extensions builtin.

```
my $result = exe $x
# or in a pipeline:
@list |> map exe |> p
```

executables

`executables` — filesystem extensions builtin.

```
my $result = executables $x
# or in a pipeline:
@list |> map executables |> p
```

exists_key

`exists_key` — batch 71 builtin.

```
my $result = exists_key $x
# or in a pipeline:
@list |> map exists_key |> p
```

exp2

`exp2` — trivial numeric / predicate builtins builtin.

```
my $result = exp2 $x
# or in a pipeline:
@list |> map exp2 |> p
```

exp_euler_step

`exp_euler_step` — batch 34 builtin.

```
my $result = exp_euler_step $x
# or in a pipeline:
@list |> map exp_euler_step |> p
```

exp_golomb_code

exp_golomb_code — batch 40 builtin.

```
my $result = exp_golomb_code $x
# or in a pipeline:
@list |> map exp_golomb_code |> p
```

exp_integral_e1

exp_integral_e1 — batch 29 builtin.

```
my $result = exp_integral_e1 $x
# or in a pipeline:
@list |> map exp_integral_e1 |> p
```

exp_smooth_double

exp_smooth_double — batch 78 builtin.

```
my $result = exp_smooth_double $x
# or in a pipeline:
@list |> map exp_smooth_double |> p
```

exp_smooth_simple

exp_smooth_simple — batch 78 builtin.

```
my $result = exp_smooth_simple $x
# or in a pipeline:
@list |> map exp_smooth_simple |> p
```

expansion_scalar_step

expansion_scalar_step — batch 39 builtin.

```
my $result = expansion_scalar_step $x
# or in a pipeline:
@list |> map expansion_scalar_step |> p
```

expected_coverage

expected_coverage — batch 33 builtin.

```
my $result = expected_coverage $x
# or in a pipeline:
@list |> map expected_coverage |> p
```

expenditure_function

expenditure_function — batch 73 builtin.

```
my $result = expenditure_function $x
# or in a pipeline:
@list |> map expenditure_function |> p
```

experience_factor

experience_factor — batch 60 builtin.

```
my $result = experience_factor $x
# or in a pipeline:
@list |> map experience_factor |> p
```

expire

`expire` — batch 71 builtin.

```
my $result = expire $x
# or in a pipeline:
@list |> map expire |> p
```

explore_exploit_epsilon

`explore_exploit_epsilon` — batch 40 builtin.

```
my $result = explore_exploit_epsilon $x
# or in a pipeline:
@list |> map explore_exploit_epsilon |> p
```

exponential_moving_average

`exponential_moving_average` — math / numeric (uncategorized batch) builtin.

```
my $result = exponential_moving_average $x
# or in a pipeline:
@list |> map exponential_moving_average |> p
```

exponential_pdf

`exponential_pdf` — math formulas builtin.

```
my $result = exponential_pdf $x
# or in a pipeline:
@list |> map exponential_pdf |> p
```

exppdf

`exppdf` — math formulas builtin. Alias for `exponential_pdf`.

```
my $result = exppdf $x
# or in a pipeline:
@list |> map exppdf |> p
```

extends

`extends` specifies parent classes for inheritance. Child classes inherit all fields and methods from parents.

```
class Animal { name: Str }
class Dog extends Animal { breed: Str }
class Hybrid extends Dog, Cat { }  # multiple inheritance
```

Inherited fields appear first in construction order. Methods are resolved child-first (override pattern).

exterior_derivative_one_form

`exterior_derivative_one_form` — batch 39 builtin.

```
my $result = exterior_derivative_one_form $x
# or in a pipeline:
@list |> map exterior_derivative_one_form |> p
```

extract_between

`extract_between` — string quote / escape builtin.

```
my $result = extract_between $x
# or in a pipeline:
@list |> map extract_between |> p
```

extrinsic_principal_curv

`extrinsic_principal_curv` — batch 39 builtin.

```
my $result = extrinsic_principal_curv $x
# or in a pipeline:
@list |> map extrinsic_principal_curv |> p
```

eye

`eye` — matrix / linear algebra builtin. Alias for `identity_matrix`.

```
my $result = eye $x
# or in a pipeline:
@list |> map eye |> p
```

eyring_k

`eyring_k` — batch 21 builtin.

```
my $result = eyring_k $x
# or in a pipeline:
@list |> map eyring_k |> p
```

eyring_rate

`eyring_rate` — batch 63 builtin.

```
my $result = eyring_rate $x
# or in a pipeline:
@list |> map eyring_rate |> p
```

f

`f` — filesystem extensions builtin.

```
my $result = f $x
# or in a pipeline:
@list |> map f |> p
```

f64_max

`f64_max` — math / numeric (uncategorized batch) builtin.

```
my $result = f64_max $x
# or in a pipeline:
@list |> map f64_max |> p
```

f64_min

`f64_min` — math / numeric (uncategorized batch) builtin.

```
my $result = f64_min $x
# or in a pipeline:
@list |> map f64_min |> p
```

f_beta

f_beta — batch 27 builtin.

```
my $result = f_beta $x
# or in a pipeline:
@list |> map f_beta |> p
```

f_pdf

f_pdf (alias fpdf) evaluates the F-distribution PDF at x with d1 and d2 degrees of freedom.

```
p fpdf(1.0, 5, 10)
```

f_statistic_pooled

f_statistic_pooled — batch 36 builtin.

```
my $result = f_statistic_pooled $x
# or in a pipeline:
@list |> map f_statistic_pooled |> p
```

f_to_c

f_to_c — unit conversions builtin.

```
my $result = f_to_c $input
```

f_to_k

f_to_k — unit conversions builtin.

```
my $result = f_to_k $input
```

face_detect_haar

face_detect_haar — batch 77 builtin.

```
my $result = face_detect_haar $x
# or in a pipeline:
@list |> map face_detect_haar |> p
```

fact

fact — trivial numeric / predicate builtins builtin. Alias for factorial.

```
my $result = fact $x
# or in a pipeline:
@list |> map fact |> p
```

factorial

factorial — trivial numeric / predicate builtins builtin.

```
my $result = factorial $x
# or in a pipeline:
@list |> map factorial |> p
```

factorial2

`factorial2` — batch 72 builtin.

```
my $result = factorial2 $x
# or in a pipeline:
@list |> map factorial2 |> p
```

factorialk

`factorialk` — batch 72 builtin.

```
my $result = factorialk $x
# or in a pipeline:
@list |> map factorialk |> p
```

faf

`faf` — stress testing builtin. Alias for `fire_and_forget`.

```
my $result = faf $x
# or in a pipeline:
@list |> map faf |> p
```

fair_division_envy_free

`fair_division_envy_free` — batch 43 builtin.

```
my $result = fair_division_envy_free $x
# or in a pipeline:
@list |> map fair_division_envy_free |> p
```

falf

`falf` — python/ruby stdlib builtin. Alias for `filterfalse`.

```
my $result = falf $x
# or in a pipeline:
@list |> map falf |> p
```

fallback

`fallback` — functional combinators builtin.

```
my $result = fallback $x
# or in a pipeline:
@list |> map fallback |> p
```

falling_factorial

`falling_factorial` — math formulas builtin.

```
my $result = falling_factorial $x
# or in a pipeline:
@list |> map falling_factorial |> p
```

false_position

`false_position` — batch 19 builtin. Alias for `regula_falsi`.

```
my $result = false_position $x
# or in a pipeline:
@list |> map false_position |> p
```

falsy_count

`falsy_count` — counters (batch 4) builtin.

```
my $result = falsy_count $x
# or in a pipeline:
@list |> map falsy_count |> p
```

fano_inequality_bound

`fano_inequality_bound` — batch 40 builtin.

```
my $result = fano_inequality_bound $x
# or in a pipeline:
@list |> map fano_inequality_bound |> p
```

faradaic_efficiency

`faradaic_efficiency` — batch 38 builtin.

```
my $result = faradaic_efficiency $x
# or in a pipeline:
@list |> map faradaic_efficiency |> p
```

faraday

`faraday_constant` (alias `faraday`) — F ≈ 96485 C/mol. Charge per mole of electrons.

```
p faraday   # 96485.33212
```

faraday_efficiency_h2

`faraday_efficiency_h2` — batch 38 builtin.

```
my $result = faraday_efficiency_h2 $x
# or in a pipeline:
@list |> map faraday_efficiency_h2 |> p
```

faraday_emf

`faraday_emf` — batch 23 builtin.

```
my $result = faraday_emf $x
# or in a pipeline:
@list |> map faraday_emf |> p
```

faraday_mass_deposited

`faraday_mass_deposited` — batch 21 builtin.

```
my $result = faraday_mass_deposited $x
# or in a pipeline:
@list |> map faraday_mass_deposited |> p
```

farmhash_64

`farmhash_64` — batch 81 builtin.

```
my $result = farmhash_64 $x
# or in a pipeline:
@list |> map farmhash_64 |> p
```

farthest_insertion_step

farthest_insertion_step — batch 41 builtin.

```
my $result = farthest_insertion_step $x
# or in a pipeline:
@list |> map farthest_insertion_step |> p
```

fb

fb — algorithms / puzzles builtin. Alias for fizzbuzz.

```
my $result = fb $x
# or in a pipeline:
@list |> map fb |> p
```

fcntl

fcntl — ipc builtin.

```
my $result = fcntl $x
# or in a pipeline:
@list |> map fcntl |> p
```

fcp

fcp — extended stdlib builtin. Alias for from_code_point.

```
my $result = fcp $x
# or in a pipeline:
@list |> map fcp |> p
```

fdivop

fdivop — extended stdlib builtin. Alias for floor_div_op.

```
my $result = fdivop $x
# or in a pipeline:
@list |> map fdivop |> p
```

feet_to_m

feet_to_m — unit conversions builtin.

```
my $result = feet_to_m $input
```

feet_to_meters

feet_to_meters — batch 30 builtin.

```
my $result = feet_to_meters $input
```

feiga

feiga — math constants builtin. Alias for feigenbaum_alpha.

```
my $result = feiga $x
# or in a pipeline:
@list |> map feiga |> p
```

feigd

`feigenbaum_delta` (alias `feigd`) — Feigenbaum constant δ ≈ 4.669. Universal constant in chaos theory.

```
p feigd   # 4.669201609102990
```

feigenbaum_alpha

`feigenbaum_alpha` — math constants builtin.

```
my $result = feigenbaum_alpha $x
# or in a pipeline:
@list |> map feigenbaum_alpha |> p
```

feldspar_classify

`feldspar_classify` — batch 65 builtin.

```
my $result = feldspar_classify $x
# or in a pipeline:
@list |> map feldspar_classify |> p
```

felsenstein_step

`felsenstein_step` — batch 33 builtin.

```
my $result = felsenstein_step $x
# or in a pipeline:
@list |> map felsenstein_step |> p
```

felzenszwalb_segment

`felzenszwalb_segment` — batch 77 builtin.

```
my $result = felzenszwalb_segment $x
# or in a pipeline:
@list |> map felzenszwalb_segment |> p
```

fenwick_build

`fenwick_build` — batch 10 builtin.

```
my $result = fenwick_build $x
# or in a pipeline:
@list |> map fenwick_build |> p
```

fenwick_for

`fenwick_for` — batch 56 builtin.

```
my $result = fenwick_for $x
# or in a pipeline:
@list |> map fenwick_for |> p
```

fenwick_query

`fenwick_query` — batch 10 builtin.

```
my $result = fenwick_query $x
# or in a pipeline:
@list |> map fenwick_query |> p
```

fermat_factor

`fermat_factor` — batch 35 builtin.

```
my $result = fermat_factor $x
# or in a pipeline:
@list |> map fermat_factor |> p
```

fermi_dirac_int

`fermi_dirac_int` — batch 29 builtin.

```
my $result = fermi_dirac_int $x
# or in a pipeline:
@list |> map fermi_dirac_int |> p
```

fermi_golden_rule

`fermi_golden_rule` — batch 32 builtin.

```
my $result = fermi_golden_rule $x
# or in a pipeline:
@list |> map fermi_golden_rule |> p
```

fermi_normal_step

`fermi_normal_step` — batch 39 builtin.

```
my $result = fermi_normal_step $x
# or in a pipeline:
@list |> map fermi_normal_step |> p
```

ferrel_cell_step

`ferrel_cell_step` — batch 42 builtin.

```
my $result = ferrel_cell_step $x
# or in a pipeline:
@list |> map ferrel_cell_step |> p
```

fetch_val

`fetch_val` — python/ruby stdlib builtin.

```
my $result = fetch_val $x
# or in a pipeline:
@list |> map fetch_val |> p
```

ffact

`ffact` — math formulas builtin. Alias for `falling_factorial`.

```
my $result = ffact $x
# or in a pipeline:
@list |> map ffact |> p
```

ffirst

`ffirst` — algebraic match builtin.

```
my $result = ffirst $x
# or in a pipeline:
@list |> map ffirst |> p
```

ffs

`ffs` — algebraic match builtin. Alias for `ffirst`.

```
my $result = ffs $x
# or in a pipeline:
@list |> map ffs |> p
```

fft_magnitude

`fft_magnitude` — math / numeric (uncategorized batch) builtin.

```
my $result = fft_magnitude $x
# or in a pipeline:
@list |> map fft_magnitude |> p
```

fftconvolve_step

`fftconvolve_step` — batch 74 builtin.

```
my $result = fftconvolve_step $x
# or in a pipeline:
@list |> map fftconvolve_step |> p
```

fftmag

`fftmag` — math / numeric (uncategorized batch) builtin. Alias for `fft_magnitude`.

```
my $result = fftmag $x
# or in a pipeline:
@list |> map fftmag |> p
```

fib

`fib` — trivial numeric / predicate builtins builtin. Alias for `fibonacci`.

```
my $result = fib $x
# or in a pipeline:
@list |> map fib |> p
```

fibonacci

`fibonacci` — trivial numeric / predicate builtins builtin.

```
my $result = fibonacci $x
# or in a pipeline:
@list |> map fibonacci |> p
```

fibonacci_code

`fibonacci_code` — batch 40 builtin.

```
my $result = fibonacci_code $x
# or in a pipeline:
@list |> map fibonacci_code |> p
```

fibonacci_seq

`fibonacci_seq` — sequences builtin.

```
my $result = fibonacci_seq $x
# or in a pipeline:
@list |> map fibonacci_seq |> p
```

fibonacci_word

`fibonacci_word` — batches 14-16 builtin.

```
my $result = fibonacci_word $x
# or in a pipeline:
@list |> map fibonacci_word |> p
```

fictitious_play_step

`fictitious_play_step` — batch 43 builtin.

```
my $result = fictitious_play_step $x
# or in a pipeline:
@list |> map fictitious_play_step |> p
```

fidelity_pure_real

`fidelity_pure_real` — batch 32 builtin.

```
my $result = fidelity_pure_real $x
# or in a pipeline:
@list |> map fidelity_pure_real |> p
```

fidx

`fidx` — python/ruby stdlib builtin. Alias for `find_index_fn`.

```
my $result = fidx $x
# or in a pipeline:
@list |> map fidx |> p
```

file_atime

`file_atime` — file stat / path builtin.

```
my $result = file_atime $x
# or in a pipeline:
@list |> map file_atime |> p
```

file_ctime

`file_ctime` — file stat / path builtin.

```
my $result = file_ctime $x
# or in a pipeline:
@list |> map file_ctime |> p
```

file_mtime

`file_mtime` — file stat / path builtin.

```
my $result = file_mtime $x
# or in a pipeline:
@list |> map file_mtime |> p
```

file_size

`file_size` — file stat / path builtin.

```
my $result = file_size $x
# or in a pipeline:
@list |> map file_size |> p
```

files

`files` — filesystem extensions builtin.

```
my $result = files $x
# or in a pipeline:
@list |> map files |> p
```

filesf

`filesf` — filesystem extensions builtin.

```
my $result = filesf $x
# or in a pipeline:
@list |> map filesf |> p
```

fill_arr

`fill_arr` — javascript array/object methods builtin.

```
my $result = fill_arr $x
# or in a pipeline:
@list |> map fill_arr |> p
```

filla

`filla` — javascript array/object methods builtin. Alias for `fill_arr`.

```
my $result = filla $x
# or in a pipeline:
@list |> map filla |> p
```

filter_bilateral

`filter_bilateral` — batch 77 builtin.

```
my $result = filter_bilateral $x
# or in a pipeline:
@list |> map filter_bilateral |> p
```

filter_chars

`filter_chars` — string helpers (batch 4) builtin.

```
my $result = filter_chars $x
# or in a pipeline:
@list |> map filter_chars |> p
```

filter_indexed

`filter_indexed` — go/general functional utilities builtin.

```
my $result = filter_indexed $x
# or in a pipeline:
@list |> map filter_indexed |> p
```

filter_map

`filter_map` — rust iterator methods builtin.

```
my $result = filter_map $x
# or in a pipeline:
@list |> map filter_map |> p
```

filter_median

`filter_median` — batch 77 builtin.

```
my $result = filter_median $x
# or in a pipeline:
@list |> map filter_median |> p
```

filter_nlmeans

`filter_nlmeans` — batch 77 builtin.

```
my $result = filter_nlmeans $x
# or in a pipeline:
@list |> map filter_nlmeans |> p
```

filter_ts

`ts_filter` — linear convolution filter. Like R's filter(method='convolution').

```
my @smooth = @{ts_filter([1,5,2,8,3], [0.25,0.5,0.25])}
```

filterfalse

`filterfalse` — python/ruby stdlib builtin.

```
my $result = filterfalse $x
# or in a pipeline:
@list |> map filterfalse |> p
```

filtfilt_pad

`filtfilt_pad` — batch 74 builtin.

```
my $result = filtfilt_pad $x
# or in a pipeline:
@list |> map filtfilt_pad |> p
```

find

`find` — functional / iterator builtin.

```
my $result = find $x
# or in a pipeline:
@list |> map find |> p
```

findInterval

find_interval — for each x, find which interval in breaks it falls into. Like R's findInterval().

```
p findInterval([1.5, 3.5, 7.5], [0, 2, 5, 10])  # [1, 2, 3]
```

find_all

find_all — functional / iterator builtin.

```
my $result = find_all $x
# or in a pipeline:
@list |> map find_all |> p
```

find_all_indices

find_all_indices — extended stdlib builtin.

```
my $result = find_all_indices $x
# or in a pipeline:
@list |> map find_all_indices |> p
```

find_first

find_first — list helpers (batch 4) builtin.

```
my $result = find_first $x
# or in a pipeline:
@list |> map find_first |> p
```

find_index

find_index — functional / iterator builtin.

```
my $result = find_index $x
# or in a pipeline:
@list |> map find_index |> p
```

find_index_fn

find_index_fn — python/ruby stdlib builtin.

```
my $result = find_index_fn $x
# or in a pipeline:
@list |> map find_index_fn |> p
```

find_indices

find_indices — additional missing stdlib functions builtin.

```
my $result = find_indices $x
# or in a pipeline:
@list |> map find_indices |> p
```

find_last

find_last — javascript array/object methods builtin.

```
my $result = find_last $x
# or in a pipeline:
@list |> map find_last |> p
```

find_last_index

`find_last_index` — javascript array/object methods builtin.

```
my $result = find_last_index $x
# or in a pipeline:
@list |> map find_last_index |> p
```

find_map

`find_map` — rust iterator methods builtin.

```
my $result = find_map $x
# or in a pipeline:
@list |> map find_map |> p
```

finite_rate_lambda

`finite_rate_lambda` — batch 22 builtin.

```
my $result = finite_rate_lambda $x
# or in a pipeline:
@list |> map finite_rate_lambda |> p
```

fintushel_stern_step

`fintushel_stern_step` — batch 37 builtin.

```
my $result = fintushel_stern_step $x
# or in a pipeline:
@list |> map fintushel_stern_step |> p
```

fip

`fip` — batch 56 builtin.

```
my $result = fip $x
# or in a pipeline:
@list |> map fip |> p
```

fir_lowpass_design

`fir_lowpass_design` — batch 25 builtin.

```
my $result = fir_lowpass_design $x
# or in a pipeline:
@list |> map fir_lowpass_design |> p
```

fir_moving_average

`fir_moving_average` — batch 25 builtin.

```
my $result = fir_moving_average $x
# or in a pipeline:
@list |> map fir_moving_average |> p
```

fire

`fire` — stress testing builtin.

```
my $result = fire $x
# or in a pipeline:
@list |> map fire |> p
```

fire_and_forget

`fire_and_forget` — stress testing builtin.

```
my $result = fire_and_forget $x
# or in a pipeline:
@list |> map fire_and_forget |> p
```

first_arg

`first_arg` — functional primitives builtin.

```
my $result = first_arg $x
# or in a pipeline:
@list |> map first_arg |> p
```

first_elem

`first_elem` — list helpers (batch 4) builtin.

```
my $result = first_elem $x
# or in a pipeline:
@list |> map first_elem |> p
```

first_eq

`first_eq` — collection (batch 2) builtin.

```
my $result = first_eq $x
# or in a pipeline:
@list |> map first_eq |> p
```

first_of_month

`first_of_month` — batch 9 builtin.

```
my $result = first_of_month $x
# or in a pipeline:
@list |> map first_of_month |> p
```

first_order_concentration

`first_order_concentration` — batch 21 builtin.

```
my $result = first_order_concentration $x
# or in a pipeline:
@list |> map first_order_concentration |> p
```

first_order_half_life

`first_order_half_life` — batch 21 builtin.

```
my $result = first_order_half_life $x
# or in a pipeline:
@list |> map first_order_half_life |> p
```

first_price_seal

first_price_seal — batch 73 builtin.

```
my $result = first_price_seal $x
# or in a pipeline:
@list |> map first_price_seal |> p
```

first_word

first_word — string (batch 2) builtin.

```
my $result = first_word $x
# or in a pipeline:
@list |> map first_word |> p
```

firwin2_freq

firwin2_freq — batch 74 builtin.

```
my $result = firwin2_freq $x
# or in a pipeline:
@list |> map firwin2_freq |> p
```

firwin_bandpass

firwin_bandpass — batch 74 builtin.

```
my $result = firwin_bandpass $x
# or in a pipeline:
@list |> map firwin_bandpass |> p
```

firwin_bandstop

firwin_bandstop — batch 74 builtin.

```
my $result = firwin_bandstop $x
# or in a pipeline:
@list |> map firwin_bandstop |> p
```

firwin_highpass

firwin_highpass — batch 74 builtin.

```
my $result = firwin_highpass $x
# or in a pipeline:
@list |> map firwin_highpass |> p
```

firwin_lowpass

firwin_lowpass — batch 74 builtin.

```
my $result = firwin_lowpass $x
# or in a pipeline:
@list |> map firwin_lowpass |> p
```

fiscal_year_us

fiscal_year_us — batch 82 builtin.

```
my $result = fiscal_year_us $x
# or in a pipeline:
@list |> map fiscal_year_us |> p
```

fisher_info_metric

`fisher_info_metric` — batch 40 builtin.

```
my $result = fisher_info_metric $x
# or in a pipeline:
@list |> map fisher_info_metric |> p
```

fisher_yates_shuffle

`fisher_yates_shuffle` — batch 9 builtin.

```
my $result = fisher_yates_shuffle $x
# or in a pipeline:
@list |> map fisher_yates_shuffle |> p
```

fista_step

`fista_step` — batch 41 builtin.

```
my $result = fista_step $x
# or in a pipeline:
@list |> map fista_step |> p
```

five_number_summary

`five_number_summary(@data)` (alias `fivenum`) — returns [min, Q1, median, Q3, max], the classic five-number summary used for box plots. Provides a quick view of data distribution.

```
my @data = 1:100
my $f = fivenum(@data)
p "min=$f->[0] Q1=$f->[1] med=$f->[2] Q3=$f->[3] max=$f->[4]"
```

fixed_from_french

`fixed_from_french` — batch 52 builtin.

```
my $result = fixed_from_french $x
# or in a pipeline:
@list |> map fixed_from_french |> p
```

fixed_from_gregorian

`fixed_from_gregorian` — batch 52 builtin.

```
my $result = fixed_from_gregorian $x
# or in a pipeline:
@list |> map fixed_from_gregorian |> p
```

fixed_from_hebrew

`fixed_from_hebrew` — batch 52 builtin.

```
my $result = fixed_from_hebrew $x
# or in a pipeline:
@list |> map fixed_from_hebrew |> p
```

fixed_from_islamic

`fixed_from_islamic` — batch 52 builtin.

```
my $result = fixed_from_islamic $x
# or in a pipeline:
@list |> map fixed_from_islamic |> p
```

fixed_from_julian

`fixed_from_julian` — batch 52 builtin.

```
my $result = fixed_from_julian $x
# or in a pipeline:
@list |> map fixed_from_julian |> p
```

fixed_from_persian

`fixed_from_persian` — batch 52 builtin.

```
my $result = fixed_from_persian $x
# or in a pipeline:
@list |> map fixed_from_persian |> p
```

fixed_quad

`fixed_quad` — batch 72 builtin.

```
my $result = fixed_quad $x
# or in a pipeline:
@list |> map fixed_quad |> p
```

fizzbuzz

`fizzbuzz` — algorithms / puzzles builtin.

```
my $result = fizzbuzz $x
# or in a pipeline:
@list |> map fizzbuzz |> p
```

flat_band_potential

`flat_band_potential` — batch 38 builtin.

```
my $result = flat_band_potential $x
# or in a pipeline:
@list |> map flat_band_potential |> p
```

flat_depth

`flat_depth` — javascript array/object methods builtin.

```
my $result = flat_depth $x
# or in a pipeline:
@list |> map flat_depth |> p
```

flat_map

`flat_map` — functional / iterator builtin.

```
my $result = flat_map $x
# or in a pipeline:
@list |> map flat_map |> p
```

flat_map_fn

`flat_map_fn` — python/ruby stdlib builtin.

```
my $result = flat_map_fn $x
# or in a pipeline:
@list |> map flat_map_fn |> p
```

flat_maps

`flat_maps { BLOCK } LIST` — a lazy streaming flat-map that evaluates the block for each element and flattens the resulting lists into a single iterator. Where `maps` expects the block to return one value, `flat_maps` handles blocks that return zero or more values per element and concatenates them seamlessly. This is the streaming equivalent of calling `map` with a multi-value block. Use it in `|>` chains when each input element fans out into multiple outputs and you want lazy evaluation.

```
1:3 |> flat_maps { (_, _ * 10) } |> e p
# 1 10 2 20 3 30
@nested |> flat_maps { @_ } |> e p   # flatten array-of-arrays
```

flatnonzero

`flatnonzero` — batch 72 builtin.

```
my $result = flatnonzero $x
# or in a pipeline:
@list |> map flatnonzero |> p
```

flatten_deep

`flatten_deep` — collection more builtin.

```
my $result = flatten_deep $x
# or in a pipeline:
@list |> map flatten_deep |> p
```

flatten_once

`flatten_once` — list helpers (batch 4) builtin.

```
my $result = flatten_once $x
# or in a pipeline:
@list |> map flatten_once |> p
```

flattop_w

`flattop_w` — batch 74 builtin.

```
my $result = flattop_w $x
# or in a pipeline:
@list |> map flattop_w |> p
```

fldr

`fldr` — rust iterator methods builtin. Alias for `fold_right`.

```
my $result = fldr $x
# or in a pipeline:
@list |> map fldr |> p
```

flesch_kincaid_grade

`flesch_kincaid_grade` — batch 9 builtin.

```
my $result = flesch_kincaid_grade $x
# or in a pipeline:
@list |> map flesch_kincaid_grade |> p
```

flesch_reading_ease

`flesch_reading_ease` — batch 9 builtin.

```
my $result = flesch_reading_ease $x
# or in a pipeline:
@list |> map flesch_reading_ease |> p
```

flip_args

`flip_args` — functional primitives builtin.

```
my $result = flip_args $x
# or in a pipeline:
@list |> map flip_args |> p
```

float_bits

`float_bits` — math functions builtin.

```
my $result = float_bits $x
# or in a pipeline:
@list |> map float_bits |> p
```

floer_homology_rank

`floer_homology_rank` — batch 37 builtin.

```
my $result = floer_homology_rank $x
# or in a pipeline:
@list |> map floer_homology_rank |> p
```

floor

`floor` — trivial numeric / predicate builtins builtin.

```
my $result = floor $x
# or in a pipeline:
@list |> map floor |> p
```

floor_div

`floor_div` — trig / math (batch 2) builtin.

```
my $result = floor_div $x
# or in a pipeline:
@list |> map floor_div |> p
```

floor_div_op

`floor_div_op` — extended stdlib builtin.

```
my $result = floor_div_op $x
# or in a pipeline:
@list |> map floor_div_op |> p
```

floor_each

`floor_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = floor_each $x
# or in a pipeline:
@list |> map floor_each |> p
```

floor_mod

`floor_mod` — extended stdlib builtin.

```
my $result = floor_mod $x
# or in a pipeline:
@list |> map floor_mod |> p
```

flow_shop_johnson_step

`flow_shop_johnson_step` — batch 41 builtin.

```
my $result = flow_shop_johnson_step $x
# or in a pipeline:
@list |> map flow_shop_johnson_step |> p
```

floyd_warshall_step

`floyd_warshall_step` — batch 75 builtin.

```
my $result = floyd_warshall_step $x
# or in a pipeline:
@list |> map floyd_warshall_step |> p
```

fltd

`fltd` — javascript array/object methods builtin. Alias for `flat_depth`.

```
my $result = fltd $x
# or in a pipeline:
@list |> map fltd |> p
```

flti

`flti` — go/general functional utilities builtin. Alias for `filter_indexed`.

```
my $result = flti $x
# or in a pipeline:
@list |> map flti |> p
```

fltm

`fltm` — rust iterator methods builtin. Alias for `filter_map`.

```
my $result = fltm $x
# or in a pipeline:
@list |> map fltm |> p
```

flux_richardson_full

`flux_richardson_full` — batch 42 builtin.

```
my $result = flux_richardson_full $x
# or in a pipeline:
@list |> map flux_richardson_full |> p
```

fma

`fma` — math functions builtin.

```
my $result = fma $x
# or in a pipeline:
@list |> map fma |> p
```

fmadd

`fmadd` — extended stdlib builtin. Alias for `fused_mul_add`.

```
my $result = fmadd $x
# or in a pipeline:
@list |> map fmadd |> p
```

fmf

`fmf` — python/ruby stdlib builtin. Alias for `flat_map_fn`.

```
my $result = fmf $x
# or in a pipeline:
@list |> map fmf |> p
```

fmin_cobyla

`fmin_cobyla` — batch 72 builtin.

```
my $result = fmin_cobyla $x
# or in a pipeline:
@list |> map fmin_cobyla |> p
```

fmin_powell

`fmin_powell` — batch 72 builtin.

```
my $result = fmin_powell $x
# or in a pipeline:
@list |> map fmin_powell |> p
```

fmod

`fmod` — extended stdlib builtin. Alias for `floor_mod`.

```
my $result = fmod $x
# or in a pipeline:
@list |> map fmod |> p
```

fndalli

`fndalli` — extended stdlib builtin. Alias for `find_all_indices`.

```
my $result = fndalli $x
# or in a pipeline:
@list |> map fndalli |> p
```

fndidxs

`fndidxs` — additional missing stdlib functions builtin. Alias for `find_indices`.

```
my $result = fndidxs $x
# or in a pipeline:
@list |> map fndidxs |> p
```

fndl

`fndl` — javascript array/object methods builtin. Alias for `find_last`.

```
my $result = fndl $x
# or in a pipeline:
@list |> map fndl |> p
```

fndli

`fndli` — javascript array/object methods builtin. Alias for `find_last_index`.

```
my $result = fndli $x
# or in a pipeline:
@list |> map fndli |> p
```

fndm

`fndm` — rust iterator methods builtin. Alias for `find_map`.

```
my $result = fndm $x
# or in a pipeline:
@list |> map fndm |> p
```

fne

`fne` — algebraic match builtin. Alias for `fnext`.

```
my $result = fne $x
# or in a pipeline:
@list |> map fne |> p
```

fnext

`fnext` — algebraic match builtin.

```
my $result = fnext $x
# or in a pipeline:
@list |> map fnext |> p
```

fnv0_32

`fnv0_32` — batch 81 builtin.

```
my $result = fnv0_32 $x
# or in a pipeline:
@list |> map fnv0_32 |> p
```

fnv1a

`fnv1a` — extended stdlib builtin.

```
my $result = fnv1a $x
# or in a pipeline:
@list |> map fnv1a |> p
```

fnv1a_32

`fnv1a_32` — batch 26 builtin.

```
my $result = fnv1a_32 $x
# or in a pipeline:
@list |> map fnv1a_32 |> p
```

fnv1a_64

`fnv1a_64` — batch 26 builtin.

```
my $result = fnv1a_64 $x
# or in a pipeline:
@list |> map fnv1a_64 |> p
```

fnv1a_hash

`fnv1a_hash` — batch 11 builtin.

```
my $result = fnv1a_hash $x
# or in a pipeline:
@list |> map fnv1a_hash |> p
```

fold_axis

`fold_axis` — batch 54 builtin.

```
my $result = fold_axis $x
# or in a pipeline:
@list |> map fold_axis |> p
```

fold_left

`fold_left` — list helpers (batch 4) builtin.

```
my $result = fold_left $x
# or in a pipeline:
@list |> map fold_left |> p
```

fold_right

`fold_right` — rust iterator methods builtin.

```
my $result = fold_right $x
# or in a pipeline:
@list |> map fold_right |> p
```

folk_theorem_value

`folk_theorem_value` — batch 43 builtin.

```
my $result = folk_theorem_value $x
# or in a pipeline:
@list |> map folk_theorem_value |> p
```

force_mass_acc

`force_mass_acc` — physics formulas builtin.

```
my $result = force_mass_acc $x
# or in a pipeline:
@list |> map force_mass_acc |> p
```

force_of_mortality

`force_of_mortality` — batch 60 builtin.

```
my $result = force_of_mortality $x
# or in a pipeline:
@list |> map force_of_mortality |> p
```

ford_fulkerson_step

`ford_fulkerson_step` — batch 75 builtin.

```
my $result = ford_fulkerson_step $x
# or in a pipeline:
@list |> map ford_fulkerson_step |> p
```

formal_charge

`formal_charge` — batch 63 builtin.

```
my $result = formal_charge $x
# or in a pipeline:
@list |> map formal_charge |> p
```

format_bytes

`format_bytes` — file stat / path builtin.

```
my $result = format_bytes $x
# or in a pipeline:
@list |> map format_bytes |> p
```

format_duration

`format_duration` — file stat / path builtin.

```
my $result = format_duration $x
# or in a pipeline:
@list |> map format_duration |> p
```

format_number

`format_number` — file stat / path builtin.

```
my $result = format_number $x
# or in a pipeline:
@list |> map format_number |> p
```

format_percent

`format_percent` — file stat / path builtin.

```
my $result = format_percent $x
# or in a pipeline:
@list |> map format_percent |> p
```

format_table_simple

format_table_simple — batch 9 builtin.

```
my $result = format_table_simple $x
# or in a pipeline:
@list |> map format_table_simple |> p
```

forward_backward_pos

forward_backward_pos — batch 69 builtin.

```
my $result = forward_backward_pos $x
# or in a pipeline:
@list |> map forward_backward_pos |> p
```

forward_kinematics_dh

forward_kinematics_dh — batch 59 builtin.

```
my $result = forward_kinematics_dh $x
# or in a pipeline:
@list |> map forward_kinematics_dh |> p
```

forward_rate

forward_rate — batch 20 builtin.

```
my $result = forward_rate $x
# or in a pipeline:
@list |> map forward_rate |> p
```

fourier_number_step

fourier_number_step — batch 42 builtin.

```
my $result = fourier_number_step $x
# or in a pipeline:
@list |> map fourier_number_step |> p
```

fp_depression

fp_depression — batch 21 builtin.

```
my $result = fp_depression $x
# or in a pipeline:
@list |> map fp_depression |> p
```

fr

fr — filesystem extensions builtin.

```
my $result = fr $x
# or in a pipeline:
@list |> map fr |> p
```

frac_part

`frac_part` — extended stdlib builtin.

```
my $result = frac_part $x
# or in a pipeline:
@list |> map frac_part |> p
```

fracp

`fracp` — extended stdlib builtin. Alias for `frac_part`.

```
my $result = fracp $x
# or in a pipeline:
@list |> map fracp |> p
```

fractional_chromatic_lower

`fractional_chromatic_lower` — batch 41 builtin.

```
my $result = fractional_chromatic_lower $x
# or in a pipeline:
@list |> map fractional_chromatic_lower |> p
```

frank_wolfe_step

`frank_wolfe_step` — batch 41 builtin.

```
my $result = frank_wolfe_step $x
# or in a pipeline:
@list |> map frank_wolfe_step |> p
```

fredkin_gate

`fredkin_gate` — batches 14-16 builtin.

```
my $result = fredkin_gate $x
# or in a pipeline:
@list |> map fredkin_gate |> p
```

free_energy_principle

`free_energy_principle` — batch 40 builtin.

```
my $result = free_energy_principle $x
# or in a pipeline:
@list |> map free_energy_principle |> p
```

free_group_rank_lower

`free_group_rank_lower` — batch 37 builtin.

```
my $result = free_group_rank_lower $x
# or in a pipeline:
@list |> map free_group_rank_lower |> p
```

free_particle_energy

`free_particle_energy` — batch 32 builtin.

```
my $result = free_particle_energy $x
# or in a pipeline:
@list |> map free_particle_energy |> p
```

freefall_time

freefall_time — batch 12 builtin.

```
my $result = freefall_time $x
# or in a pipeline:
@list |> map freefall_time |> p
```

freefall_velocity_schwarzschild

freefall_velocity_schwarzschild — batch 31 builtin.

```
my $result = freefall_velocity_schwarzschild $x
# or in a pipeline:
@list |> map freefall_velocity_schwarzschild |> p
```

french_revolutionary_leap

french_revolutionary_leap — batch 52 builtin.

```
my $result = french_revolutionary_leap $x
# or in a pipeline:
@list |> map french_revolutionary_leap |> p
```

freq_wavelength

freq_wavelength — physics formulas builtin.

```
my $result = freq_wavelength $x
# or in a pipeline:
@list |> map freq_wavelength |> p
```

freqs_eval

freqs_eval — batch 74 builtin.

```
my $result = freqs_eval $x
# or in a pipeline:
@list |> map freqs_eval |> p
```

freqz_eval

freqz_eval — batch 74 builtin.

```
my $result = freqz_eval $x
# or in a pipeline:
@list |> map freqz_eval |> p
```

fresnel_reflection_normal

fresnel_reflection_normal — batch 23 builtin.

```
my $result = fresnel_reflection_normal $x
# or in a pipeline:
@list |> map fresnel_reflection_normal |> p
```

fresnel_rp

fresnel_rp — batch 23 builtin.

```
my $result = fresnel_rp $x
# or in a pipeline:
@list |> map fresnel_rp |> p
```

fresnel_rs

`fresnel_rs` — batch 23 builtin.

```
my $result = fresnel_rs $x
# or in a pipeline:
@list |> map fresnel_rs |> p
```

friction_factor_laminar

`friction_factor_laminar` — batch 13 builtin.

```
my $result = friction_factor_laminar $x
# or in a pipeline:
@list |> map friction_factor_laminar |> p
```

friedmann_density_total

`friedmann_density_total` — batch 31 builtin.

```
my $result = friedmann_density_total $x
# or in a pipeline:
@list |> map friedmann_density_total |> p
```

from_base

`from_base` — base conversion builtin.

```
my $result = from_base $x
# or in a pipeline:
@list |> map from_base |> p
```

from_bin

`from_bin` — base conversion builtin.

```
my $result = from_bin $x
# or in a pipeline:
@list |> map from_bin |> p
```

from_code_point

`from_code_point` — extended stdlib builtin.

```
my $result = from_code_point $x
# or in a pipeline:
@list |> map from_code_point |> p
```

from_csv_line

`from_csv_line` — string helpers (batch 4) builtin.

```
my $result = from_csv_line $x
# or in a pipeline:
@list |> map from_csv_line |> p
```

from_digits

`from_digits` — list helpers (batch 4) builtin.

```
my $result = from_digits $x
# or in a pipeline:
@list |> map from_digits |> p
```

from_hex

`from_hex` — base conversion builtin.

```
my $result = from_hex $x
# or in a pipeline:
@list |> map from_hex |> p
```

from_oct

`from_oct` — base conversion builtin.

```
my $result = from_oct $x
# or in a pipeline:
@list |> map from_oct |> p
```

from_pairs

`from_pairs` — list helpers (batch 4) builtin.

```
my $result = from_pairs $x
# or in a pipeline:
@list |> map from_pairs |> p
```

froude_number

`froude_number` — batch 13 builtin.

```
my $result = froude_number $x
# or in a pipeline:
@list |> map froude_number |> p
```

froude_number_step

`froude_number_step` — batch 42 builtin.

```
my $result = froude_number_step $x
# or in a pipeline:
@list |> map froude_number_step |> p
```

frustum_volume

`frustum_volume($r1, $r2, $h)` — computes the volume of a conical frustum (truncated cone) with base radii r1 and r2 and height h.

```
p frustum_volume(5, 3, 10)   # ~513
```

frustvol

`frustvol` — geometry (extended) builtin. Alias for `frustum_volume`.

```
my $result = frustvol $x
# or in a pipeline:
@list |> map frustvol |> p
```

fsize

`fsize` — file stat / path builtin. Alias for `file_size`.

```
my $result = fsize $x
# or in a pipeline:
@list |> map fsize |> p
```

fst

`fst` — functional / iterator builtin. Alias for `first`.

```
my $result = fst $x
# or in a pipeline:
@list |> map fst |> p
```

ft

`ft` — data / network builtin. Alias for `fetch`.

```
my $result = ft $x
# or in a pipeline:
@list |> map ft |> p
```

fta

`fta` — data / network builtin. Alias for `fetch_async`.

```
my $result = fta $x
# or in a pipeline:
@list |> map fta |> p
```

ftaj

`ftaj` — data / network builtin. Alias for `fetch_async_json`.

```
my $result = ftaj $x
# or in a pipeline:
@list |> map ftaj |> p
```

ftj

`ftj` — data / network builtin. Alias for `fetch_json`.

```
my $result = ftj $x
# or in a pipeline:
@list |> map ftj |> p
```

fuel_cell_polarization

`fuel_cell_polarization` — batch 38 builtin.

```
my $result = fuel_cell_polarization $x
# or in a pipeline:
@list |> map fuel_cell_polarization |> p
```

fundamental_group_zn

`fundamental_group_zn` — batch 37 builtin.

```
my $result = fundamental_group_zn $x
# or in a pipeline:
@list |> map fundamental_group_zn |> p
```

fused_mul_add

`fused_mul_add` — extended stdlib builtin.

```
my $result = fused_mul_add $x
# or in a pipeline:
@list |> map fused_mul_add |> p
```

fusion_rule_su2_step

`fusion_rule_su2_step` — batch 37 builtin.

```
my $result = fusion_rule_su2_step $x
# or in a pipeline:
@list |> map fusion_rule_su2_step |> p
```

future_value

`future_value` — physics formulas builtin.

```
my $result = future_value $x
# or in a pipeline:
@list |> map future_value |> p
```

fv

`fv` — python/ruby stdlib builtin. Alias for `fetch_val`.

```
my $result = fv $x
# or in a pipeline:
@list |> map fv |> p
```

fx_forward

`fx_forward` — batch 20 builtin.

```
my $result = fx_forward $x
# or in a pipeline:
@list |> map fx_forward |> p
```

g2_zero

`g2_zero` — batch 32 builtin.

```
my $result = g2_zero $x
# or in a pipeline:
@list |> map g2_zero |> p
```

g_to_oz

`g_to_oz` — unit conversions builtin.

```
my $result = g_to_oz $input
```

gabor_filter

`gabor_filter` — batch 77 builtin.

```
my $result = gabor_filter $x
# or in a pipeline:
@list |> map gabor_filter |> p
```

gabor_kernel

`gabor_kernel` — batches 14-16 builtin.

```
my $result = gabor_kernel $x
# or in a pipeline:
@list |> map gabor_kernel |> p
```

gae_advantage_step

`gae_advantage_step` — batch 40 builtin.

```
my $result = gae_advantage_step $x
# or in a pipeline:
@list |> map gae_advantage_step |> p
```

gain_ratio

`gain_ratio` — batch 27 builtin.

```
my $result = gain_ratio $x
# or in a pipeline:
@list |> map gain_ratio |> p
```

gale_optimal

`gale_optimal` — batch 73 builtin.

```
my $result = gale_optimal $x
# or in a pipeline:
@list |> map gale_optimal |> p
```

gale_shapley_step

`gale_shapley_step` — batch 73 builtin.

```
my $result = gale_shapley_step $x
# or in a pipeline:
@list |> map gale_shapley_step |> p
```

gallons_to_liters

`gallons_to_liters` — file stat / path builtin.

```
my $result = gallons_to_liters $input
```

game_of_life_step

`game_of_life_step` — algorithms / puzzles builtin.

```
my $result = game_of_life_step $x
# or in a pipeline:
@list |> map game_of_life_step |> p
```

game_two_player_value

`game_two_player_value` — batch 43 builtin.

```
my $result = game_two_player_value $x
# or in a pipeline:
@list |> map game_two_player_value |> p
```

gamma_approx

`gamma_approx($z)` — Gamma function Γ(z) using Lanczos approximation. Extends factorial: Γ(n) = (n-1)!

```
p gamma(5)    # 24 (same as 4!)
p gamma(0.5)  # ~1.772 (√π)
```

gamma_pdf

`gamma_pdf` (alias `gammapdf`) evaluates the Gamma distribution PDF at x with shape k and scale theta.

```
p gammapdf(2.0, 2, 1)  # Gamma(2,1) at x=2
```

gamow_factor

`gamow_factor` — batch 32 builtin.

```
my $result = gamow_factor $x
# or in a pipeline:
@list |> map gamow_factor |> p
```

garch_fit

`garch_fit` — batch 78 builtin.

```
my $result = garch_fit $x
# or in a pipeline:
@list |> map garch_fit |> p
```

garch_step

`garch_step` — batches 17-18 builtin.

```
my $result = garch_step $x
# or in a pipeline:
@list |> map garch_step |> p
```

garman_kohlhagen_call

`garman_kohlhagen_call` — batch 20 builtin.

```
my $result = garman_kohlhagen_call $x
# or in a pipeline:
@list |> map garman_kohlhagen_call |> p
```

gas_constant

`gas_constant` (alias `rgas`) — R ≈ 8.314 J/(mol·K). Universal gas constant.

```
p rgas   # 8.314462618
```

gas_constant_r

`gas_constant_r` — batch 12 builtin.

```
my $result = gas_constant_r $x
# or in a pipeline:
@list |> map gas_constant_r |> p
```

gas_constant_value

`gas_constant_value` — batch 63 builtin.

```
my $result = gas_constant_value $x
# or in a pipeline:
@list |> map gas_constant_value |> p
```

gasp

`ideal_gas_pressure($n, $V, $T)` (alias `gasp`) — pressure P = nRT/V. n moles, V volume (m³), T temperature (K).

```
p gasp(1, 0.0224, 273.15)   # ~101325 Pa (1 mol at STP)
```

gasv

`gasv` — physics formulas builtin. Alias for `ideal_gas_volume`.

```
my $result = gasv $x
# or in a pipeline:
@list |> map gasv |> p
```

gate_decompose

`gate_decompose` — batch 80 builtin.

```
my $result = gate_decompose $x
# or in a pipeline:
@list |> map gate_decompose |> p
```

gauss_bonnet_term_2d

`gauss_bonnet_term_2d` — batch 39 builtin.

```
my $result = gauss_bonnet_term_2d $x
# or in a pipeline:
@list |> map gauss_bonnet_term_2d |> p
```

gauss_bonnet_total

`gauss_bonnet_total` — batch 37 builtin.

```
my $result = gauss_bonnet_total $x
# or in a pipeline:
@list |> map gauss_bonnet_total |> p
```

gauss_codazzi_step

`gauss_codazzi_step` — batch 39 builtin.

```
my $result = gauss_codazzi_step $x
# or in a pipeline:
@list |> map gauss_codazzi_step |> p
```

gauss_irk_2_stage

`gauss_irk_2_stage` — batch 34 builtin.

```
my $result = gauss_irk_2_stage $x
# or in a pipeline:
@list |> map gauss_irk_2_stage |> p
```

gauss_kronrod_15

`gauss_kronrod_15` — batch 19 builtin.

```
my $result = gauss_kronrod_15 $x
# or in a pipeline:
@list |> map gauss_kronrod_15 |> p
```

gauss_legendre_5

`gauss_legendre_5` — batch 19 builtin.

```
my $result = gauss_legendre_5 $x
# or in a pipeline:
@list |> map gauss_legendre_5 |> p
```

gaussian_curvature_step

`gaussian_curvature_step` — batch 39 builtin.

```
my $result = gaussian_curvature_step $x
# or in a pipeline:
@list |> map gaussian_curvature_step |> p
```

gaussian_filter_window

`gaussian_filter_window` — batch 40 builtin.

```
my $result = gaussian_filter_window $x
# or in a pipeline:
@list |> map gaussian_filter_window |> p
```

gaussian_window

`gaussian_window` — batch 25 builtin.

```
my $result = gaussian_window $x
# or in a pipeline:
@list |> map gaussian_window |> p
```

gb_to_bytes

`gb_to_bytes` — unit conversions builtin.

```
my $result = gb_to_bytes $input
```

gbf

`gbf` — go/general functional utilities builtin. Alias for `group_by_fn`.

```
my $result = gbf $x
# or in a pipeline:
@list |> map gbf |> p
```

gbi

`gbi` — python/ruby stdlib builtin. Alias for `groupby_iter`.

```
my $result = gbi $x
# or in a pipeline:
@list |> map gbi |> p
```

gcby

`gcby` — matrix operations (uncategorized batch) builtin. Alias for `group_consecutive_by`.

```
my $result = gcby $x
# or in a pipeline:
@list |> map gcby |> p
```

gcd

`gcd` — trivial numeric / predicate builtins builtin.

```
my $result = gcd $x
# or in a pipeline:
@list |> map gcd |> p
```

gconst

`gconst` — constants builtin. Alias for `gravitational_constant`.

```
my $result = gconst $x
# or in a pipeline:
@list |> map gconst |> p
```

gelu

`gelu` applies the Gaussian Error Linear Unit, used in BERT/GPT transformers.

```
p gelu(1)    # 0.8412
p gelu(-1)   # -0.1588
```

generalized_advantage

`generalized_advantage` — batch 40 builtin.

```
my $result = generalized_advantage $x
# or in a pipeline:
@list |> map generalized_advantage |> p
```

generation_time

`generation_time` — batch 22 builtin.

```
my $result = generation_time $x
# or in a pipeline:
@list |> map generation_time |> p
```

generation_time_demo

`generation_time_demo` — batch 22 builtin.

```
my $result = generation_time_demo $x
# or in a pipeline:
@list |> map generation_time_demo |> p
```

generation_time_step

`generation_time_step` — batch 61 builtin.

```
my $result = generation_time_step $x
# or in a pipeline:
@list |> map generation_time_step |> p
```

genetic_crossover_one_point

genetic_crossover_one_point — batch 41 builtin.

```
my $result = genetic_crossover_one_point $x
# or in a pipeline:
@list |> map genetic_crossover_one_point |> p
```

genus_curve_arith

genus_curve_arith — batch 37 builtin.

```
my $result = genus_curve_arith $x
# or in a pipeline:
@list |> map genus_curve_arith |> p
```

genus_curve_geo

genus_curve_geo — batch 37 builtin.

```
my $result = genus_curve_geo $x
# or in a pipeline:
@list |> map genus_curve_geo |> p
```

genus_from_euler

genus_from_euler — batch 28 builtin.

```
my $result = genus_from_euler $x
# or in a pipeline:
@list |> map genus_from_euler |> p
```

genus_surface

genus_surface — batch 37 builtin.

```
my $result = genus_surface $x
# or in a pipeline:
@list |> map genus_surface |> p
```

geoadd

geoadd — batch 71 builtin.

```
my $result = geoadd $x
# or in a pipeline:
@list |> map geoadd |> p
```

geodesic_curvature_step

geodesic_curvature_step — batch 39 builtin.

```
my $result = geodesic_curvature_step $x
# or in a pipeline:
@list |> map geodesic_curvature_step |> p
```

geodesic_equation_step_zero

geodesic_equation_step_zero — batch 39 builtin.

```
my $result = geodesic_equation_step_zero $x
# or in a pipeline:
@list |> map geodesic_equation_step_zero |> p
```

geodist

`geodist` — batch 71 builtin.

```
my $result = geodist $x
# or in a pipeline:
@list |> map geodist |> p
```

geohash

`geohash` — batch 71 builtin.

```
my $result = geohash $x
# or in a pipeline:
@list |> map geohash |> p
```

geohash_bbox

`geohash_bbox` — batches 14-16 builtin.

```
my $result = geohash_bbox $x
# or in a pipeline:
@list |> map geohash_bbox |> p
```

geohash_decode

`geohash_decode` — batches 14-16 builtin.

```
my $result = geohash_decode $x
# or in a pipeline:
@list |> map geohash_decode |> p
```

geohash_encode

`geohash_encode` — batches 14-16 builtin.

```
my $result = geohash_encode $x
# or in a pipeline:
@list |> map geohash_encode |> p
```

geohash_neighbor

`geohash_neighbor` — batches 14-16 builtin.

```
my $result = geohash_neighbor $x
# or in a pipeline:
@list |> map geohash_neighbor |> p
```

geohash_neighbors

`geohash_neighbors` — batch 58 builtin.

```
my $result = geohash_neighbors $x
# or in a pipeline:
@list |> map geohash_neighbors |> p
```

geomagnetic_kp_index

`geomagnetic_kp_index` — batch 42 builtin.

```
my $result = geomagnetic_kp_index $x
# or in a pipeline:
@list |> map geomagnetic_kp_index |> p
```

geometric_height_full

`geometric_height_full` — batch 42 builtin.

```
my $result = geometric_height_full $x
# or in a pipeline:
@list |> map geometric_height_full |> p
```

geometric_intersection_number

`geometric_intersection_number` — batch 37 builtin.

```
my $result = geometric_intersection_number $x
# or in a pipeline:
@list |> map geometric_intersection_number |> p
```

geometric_mean

`geometric_mean` — file stat / path builtin.

```
my $result = geometric_mean $x
# or in a pipeline:
@list |> map geometric_mean |> p
```

geometric_mean_arr

`geometric_mean_arr` — batch 11 builtin.

```
my $result = geometric_mean_arr $x
# or in a pipeline:
@list |> map geometric_mean_arr |> p
```

geometric_sequence

`geometric_sequence` — batch 11 builtin.

```
my $result = geometric_sequence $x
# or in a pipeline:
@list |> map geometric_sequence |> p
```

geometric_series

`geometric_series($a1, $r, $n)` (alias `geomser`) — sum of n terms of geometric sequence. Formula: a1×(1-r^n)/(1-r). If r=1, returns a1×n.

```
p geomser(1, 2, 10)      # 1023 (1+2+4+...+512)
p geomser(1, 0.5, 10)    # ~1.998 (converging to 2)
```

geopotential_height_full

`geopotential_height_full` — batch 42 builtin.

```
my $result = geopotential_height_full $x
# or in a pipeline:
@list |> map geopotential_height_full |> p
```

geostrophic_wind_step

`geostrophic_wind_step` — batch 42 builtin.

```
my $result = geostrophic_wind_step $x
# or in a pipeline:
@list |> map geostrophic_wind_step |> p
```

get_in

`get_in` — algebraic match builtin.

```
my $result = get_in $x
# or in a pipeline:
@list |> map get_in |> p
```

gethostent

`gethostent` — posix metadata builtin.

```
my $result = gethostent $x
# or in a pipeline:
@list |> map gethostent |> p
```

getnetbyname

`getnetbyname` — posix metadata builtin.

```
my $result = getnetbyname $x
# or in a pipeline:
@list |> map getnetbyname |> p
```

getnetent

`getnetent` — posix metadata builtin.

```
my $result = getnetent $x
# or in a pipeline:
@list |> map getnetent |> p
```

getprotoent

`getprotoent` — posix metadata builtin.

```
my $result = getprotoent $x
# or in a pipeline:
@list |> map getprotoent |> p
```

getrange

`getrange` — batch 71 builtin.

```
my $result = getrange $x
# or in a pipeline:
@list |> map getrange |> p
```

getservent

`getservent` — posix metadata builtin.

```
my $result = getservent $x
# or in a pipeline:
@list |> map getservent |> p
```

getset

`getset` — batch 71 builtin.

```
my $result = getset $x
# or in a pipeline:
@list |> map getset |> p
```

gf256_multiply

`gf256_multiply` — batch 35 builtin.

```
my $result = gf256_multiply $x
# or in a pipeline:
@list |> map gf256_multiply |> p
```

gforce

`gravitational_force($m1, $m2, $r)` (alias `gforce`) — Newton's law of gravitation: $F = Gm_1m_2/r^2$.

```
p gforce(5.972e24, 1000, 6.371e6)   # ~9820 N (1 ton at Earth surface)
```

gfx_aabb_intersect_check

`gfx_aabb_intersect_check` — batch 47 builtin.

```
my $result = gfx_aabb_intersect_check $x
# or in a pipeline:
@list |> map gfx_aabb_intersect_check |> p
```

gfx_axis_angle_to_quat

`gfx_axis_angle_to_quat` — batch 47 builtin.

```
my $result = gfx_axis_angle_to_quat $input
```

gfx_barycentric_alpha

`gfx_barycentric_alpha` — batch 47 builtin.

```
my $result = gfx_barycentric_alpha $x
# or in a pipeline:
@list |> map gfx_barycentric_alpha |> p
```

gfx_barycentric_beta

`gfx_barycentric_beta` — batch 47 builtin.

```
my $result = gfx_barycentric_beta $x
# or in a pipeline:
@list |> map gfx_barycentric_beta |> p
```

gfx_barycentric_gamma

`gfx_barycentric_gamma` — batch 47 builtin.

```
my $result = gfx_barycentric_gamma $x
# or in a pipeline:
@list |> map gfx_barycentric_gamma |> p
```

gfx_blinn_specular_step

gfx_blinn_specular_step — batch 47 builtin.

```
my $result = gfx_blinn_specular_step $x
# or in a pipeline:
@list |> map gfx_blinn_specular_step |> p
```

gfx_blue_noise_value

gfx_blue_noise_value — batch 47 builtin.

```
my $result = gfx_blue_noise_value $x
# or in a pipeline:
@list |> map gfx_blue_noise_value |> p
```

gfx_bresenham_step_x

gfx_bresenham_step_x — batch 47 builtin.

```
my $result = gfx_bresenham_step_x $x
# or in a pipeline:
@list |> map gfx_bresenham_step_x |> p
```

gfx_bresenham_step_y

gfx_bresenham_step_y — batch 47 builtin.

```
my $result = gfx_bresenham_step_y $x
# or in a pipeline:
@list |> map gfx_bresenham_step_y |> p
```

gfx_clip_polygon_step

gfx_clip_polygon_step — batch 47 builtin.

```
my $result = gfx_clip_polygon_step $x
# or in a pipeline:
@list |> map gfx_clip_polygon_step |> p
```

gfx_cohen_sutherland_code

gfx_cohen_sutherland_code — batch 47 builtin.

```
my $result = gfx_cohen_sutherland_code $x
# or in a pipeline:
@list |> map gfx_cohen_sutherland_code |> p
```

gfx_cook_torrance_d_ggx

gfx_cook_torrance_d_ggx — batch 47 builtin.

```
my $result = gfx_cook_torrance_d_ggx $x
# or in a pipeline:
@list |> map gfx_cook_torrance_d_ggx |> p
```

gfx_cook_torrance_f_schlick

gfx_cook_torrance_f_schlick — batch 47 builtin.

```
my $result = gfx_cook_torrance_f_schlick $x
# or in a pipeline:
@list |> map gfx_cook_torrance_f_schlick |> p
```

gfx_cook_torrance_g_smith

gfx_cook_torrance_g_smith — batch 47 builtin.

```
my $result = gfx_cook_torrance_g_smith $x
# or in a pipeline:
@list |> map gfx_cook_torrance_g_smith |> p
```

gfx_cube_map_face_index

gfx_cube_map_face_index — batch 47 builtin.

```
my $result = gfx_cube_map_face_index $x
# or in a pipeline:
@list |> map gfx_cube_map_face_index |> p
```

gfx_curl_noise_step

gfx_curl_noise_step — batch 47 builtin.

```
my $result = gfx_curl_noise_step $x
# or in a pipeline:
@list |> map gfx_curl_noise_step |> p
```

gfx_disney_principled_d

gfx_disney_principled_d — batch 47 builtin.

```
my $result = gfx_disney_principled_d $x
# or in a pipeline:
@list |> map gfx_disney_principled_d |> p
```

gfx_dither_bayer_4x4

gfx_dither_bayer_4x4 — batch 47 builtin.

```
my $result = gfx_dither_bayer_4x4 $x
# or in a pipeline:
@list |> map gfx_dither_bayer_4x4 |> p
```

gfx_dither_floyd_steinberg

gfx_dither_floyd_steinberg — batch 47 builtin.

```
my $result = gfx_dither_floyd_steinberg $x
# or in a pipeline:
@list |> map gfx_dither_floyd_steinberg |> p
```

gfx_environment_map_uv_u

gfx_environment_map_uv_u — batch 47 builtin.

```
my $result = gfx_environment_map_uv_u $x
# or in a pipeline:
@list |> map gfx_environment_map_uv_u |> p
```

gfx_environment_map_uv_v

gfx_environment_map_uv_v — batch 47 builtin.

```
my $result = gfx_environment_map_uv_v $x
# or in a pipeline:
@list |> map gfx_environment_map_uv_v |> p
```

gfx_euler_to_quat_w

gfx_euler_to_quat_w — batch 47 builtin.

```
my $result = gfx_euler_to_quat_w $input
```

gfx_euler_to_quat_x

gfx_euler_to_quat_x — batch 47 builtin.

```
my $result = gfx_euler_to_quat_x $input
```

gfx_euler_to_quat_y

gfx_euler_to_quat_y — batch 47 builtin.

```
my $result = gfx_euler_to_quat_y $input
```

gfx_euler_to_quat_z

gfx_euler_to_quat_z — batch 47 builtin.

```
my $result = gfx_euler_to_quat_z $input
```

gfx_fbm_noise_step

gfx_fbm_noise_step — batch 47 builtin.

```
my $result = gfx_fbm_noise_step $x
# or in a pipeline:
@list |> map gfx_fbm_noise_step |> p
```

gfx_fresnel_conductor_step

gfx_fresnel_conductor_step — batch 47 builtin.

```
my $result = gfx_fresnel_conductor_step $x
# or in a pipeline:
@list |> map gfx_fresnel_conductor_step |> p
```

gfx_fresnel_dielectric_step

gfx_fresnel_dielectric_step — batch 47 builtin.

```
my $result = gfx_fresnel_dielectric_step $x
# or in a pipeline:
@list |> map gfx_fresnel_dielectric_step |> p
```

gfx_gamma_correct_step

gfx_gamma_correct_step — batch 47 builtin.

```
my $result = gfx_gamma_correct_step $x
# or in a pipeline:
@list |> map gfx_gamma_correct_step |> p
```

gfx_geometric_attenuation_smith

gfx_geometric_attenuation_smith — batch 47 builtin.

```
my $result = gfx_geometric_attenuation_smith $x
# or in a pipeline:
@list |> map gfx_geometric_attenuation_smith |> p
```

gfx_gradient_noise_step

`gfx_gradient_noise_step` — batch 47 builtin.

```
my $result = gfx_gradient_noise_step $x
# or in a pipeline:
@list |> map gfx_gradient_noise_step |> p
```

gfx_halton_step

`gfx_halton_step` — batch 47 builtin.

```
my $result = gfx_halton_step $x
# or in a pipeline:
@list |> map gfx_halton_step |> p
```

gfx_homogeneous_divide

`gfx_homogeneous_divide` — batch 47 builtin.

```
my $result = gfx_homogeneous_divide $x
# or in a pipeline:
@list |> map gfx_homogeneous_divide |> p
```

gfx_index_of_refraction

`gfx_index_of_refraction` — batch 47 builtin.

```
my $result = gfx_index_of_refraction $x
# or in a pipeline:
@list |> map gfx_index_of_refraction |> p
```

gfx_irradiance_sh_eval

`gfx_irradiance_sh_eval` — batch 47 builtin.

```
my $result = gfx_irradiance_sh_eval $x
# or in a pipeline:
@list |> map gfx_irradiance_sh_eval |> p
```

gfx_lambert_term

`gfx_lambert_term` — batch 47 builtin.

```
my $result = gfx_lambert_term $x
# or in a pipeline:
@list |> map gfx_lambert_term |> p
```

gfx_liang_barsky_t

`gfx_liang_barsky_t` — batch 47 builtin.

```
my $result = gfx_liang_barsky_t $x
# or in a pipeline:
@list |> map gfx_liang_barsky_t |> p
```

gfx_linear_to_srgb

gfx_linear_to_srgb — batch 47 builtin.

```
my $result = gfx_linear_to_srgb $input
```

gfx_lookat_forward

gfx_lookat_forward — batch 47 builtin.

```
my $result = gfx_lookat_forward $x
# or in a pipeline:
@list |> map gfx_lookat_forward |> p
```

gfx_lookat_right

gfx_lookat_right — batch 47 builtin.

```
my $result = gfx_lookat_right $x
# or in a pipeline:
@list |> map gfx_lookat_right |> p
```

gfx_lookat_up

gfx_lookat_up — batch 47 builtin.

```
my $result = gfx_lookat_up $x
# or in a pipeline:
@list |> map gfx_lookat_up |> p
```

gfx_low_discrepancy_step

gfx_low_discrepancy_step — batch 47 builtin.

```
my $result = gfx_low_discrepancy_step $x
# or in a pipeline:
@list |> map gfx_low_discrepancy_step |> p
```

gfx_microfacet_brdf_step

gfx_microfacet_brdf_step — batch 47 builtin.

```
my $result = gfx_microfacet_brdf_step $x
# or in a pipeline:
@list |> map gfx_microfacet_brdf_step |> p
```

gfx_ndc_to_screen_x

gfx_ndc_to_screen_x — batch 47 builtin.

```
my $result = gfx_ndc_to_screen_x $input
```

gfx_ndc_to_screen_y

gfx_ndc_to_screen_y — batch 47 builtin.

```
my $result = gfx_ndc_to_screen_y $input
```

gfx_normal_distribution_ggx

gfx_normal_distribution_ggx — batch 47 builtin.

```
my $result = gfx_normal_distribution_ggx $x
# or in a pipeline:
@list |> map gfx_normal_distribution_ggx |> p
```

gfx_obb_overlap_step

gfx_obb_overlap_step — batch 47 builtin.

```
my $result = gfx_obb_overlap_step $x
# or in a pipeline:
@list |> map gfx_obb_overlap_step |> p
```

gfx_octahedral_encode_x

gfx_octahedral_encode_x — batch 47 builtin.

```
my $result = gfx_octahedral_encode_x $x
# or in a pipeline:
@list |> map gfx_octahedral_encode_x |> p
```

gfx_octahedral_encode_y

gfx_octahedral_encode_y — batch 47 builtin.

```
my $result = gfx_octahedral_encode_y $x
# or in a pipeline:
@list |> map gfx_octahedral_encode_y |> p
```

gfx_oklab_a_step

gfx_oklab_a_step — batch 47 builtin.

```
my $result = gfx_oklab_a_step $x
# or in a pipeline:
@list |> map gfx_oklab_a_step |> p
```

gfx_oklab_b_step

gfx_oklab_b_step — batch 47 builtin.

```
my $result = gfx_oklab_b_step $x
# or in a pipeline:
@list |> map gfx_oklab_b_step |> p
```

gfx_oklab_l_step

gfx_oklab_l_step — batch 47 builtin.

```
my $result = gfx_oklab_l_step $x
# or in a pipeline:
@list |> map gfx_oklab_l_step |> p
```

gfx_oklch_chroma

gfx_oklch_chroma — batch 47 builtin.

```
my $result = gfx_oklch_chroma $x
# or in a pipeline:
@list |> map gfx_oklch_chroma |> p
```

gfx_oklch_hue

gfx_oklch_hue — batch 47 builtin.

```
my $result = gfx_oklch_hue $x
# or in a pipeline:
@list |> map gfx_oklch_hue |> p
```

gfx_oren_nayar_term

gfx_oren_nayar_term — batch 47 builtin.

```
my $result = gfx_oren_nayar_term $x
# or in a pipeline:
@list |> map gfx_oren_nayar_term |> p
```

gfx_orthographic_proj

gfx_orthographic_proj — batch 47 builtin.

```
my $result = gfx_orthographic_proj $x
# or in a pipeline:
@list |> map gfx_orthographic_proj |> p
```

gfx_pcg_hash_step

gfx_pcg_hash_step — batch 47 builtin.

```
my $result = gfx_pcg_hash_step $x
# or in a pipeline:
@list |> map gfx_pcg_hash_step |> p
```

gfx_perlin_noise_step

gfx_perlin_noise_step — batch 47 builtin.

```
my $result = gfx_perlin_noise_step $x
# or in a pipeline:
@list |> map gfx_perlin_noise_step |> p
```

gfx_perspective_proj_x

gfx_perspective_proj_x — batch 47 builtin.

```
my $result = gfx_perspective_proj_x $x
# or in a pipeline:
@list |> map gfx_perspective_proj_x |> p
```

gfx_perspective_proj_y

gfx_perspective_proj_y — batch 47 builtin.

```
my $result = gfx_perspective_proj_y $x
# or in a pipeline:
@list |> map gfx_perspective_proj_y |> p
```

gfx_phong_ambient_step

gfx_phong_ambient_step — batch 47 builtin.

```
my $result = gfx_phong_ambient_step $x
# or in a pipeline:
@list |> map gfx_phong_ambient_step |> p
```

gfx_phong_diffuse_step

`gfx_phong_diffuse_step` — batch 47 builtin.

```
my $result = gfx_phong_diffuse_step $x
# or in a pipeline:
@list |> map gfx_phong_diffuse_step |> p
```

gfx_phong_specular_step

`gfx_phong_specular_step` — batch 47 builtin.

```
my $result = gfx_phong_specular_step $x
# or in a pipeline:
@list |> map gfx_phong_specular_step |> p
```

gfx_quat_dot_two

`gfx_quat_dot_two` — batch 47 builtin.

```
my $result = gfx_quat_dot_two $x
# or in a pipeline:
@list |> map gfx_quat_dot_two |> p
```

gfx_quat_inverse_step

`gfx_quat_inverse_step` — batch 47 builtin.

```
my $result = gfx_quat_inverse_step $x
# or in a pipeline:
@list |> map gfx_quat_inverse_step |> p
```

gfx_quat_nlerp_step

`gfx_quat_nlerp_step` — batch 47 builtin.

```
my $result = gfx_quat_nlerp_step $x
# or in a pipeline:
@list |> map gfx_quat_nlerp_step |> p
```

gfx_quat_slerp_step

`gfx_quat_slerp_step` — batch 47 builtin.

```
my $result = gfx_quat_slerp_step $x
# or in a pipeline:
@list |> map gfx_quat_slerp_step |> p
```

gfx_quat_to_axis_angle

`gfx_quat_to_axis_angle` — batch 47 builtin.

```
my $result = gfx_quat_to_axis_angle $input
```

gfx_quat_to_euler_pitch

`gfx_quat_to_euler_pitch` — batch 47 builtin.

```
my $result = gfx_quat_to_euler_pitch $input
```

gfx_quat_to_euler_roll

gfx_quat_to_euler_roll — batch 47 builtin.

```
my $result = gfx_quat_to_euler_roll $input
```

gfx_quat_to_euler_yaw

gfx_quat_to_euler_yaw — batch 47 builtin.

```
my $result = gfx_quat_to_euler_yaw $input
```

gfx_radiance_sh_eval

gfx_radiance_sh_eval — batch 47 builtin.

```
my $result = gfx_radiance_sh_eval $x
# or in a pipeline:
@list |> map gfx_radiance_sh_eval |> p
```

gfx_ray_box_t

gfx_ray_box_t — batch 47 builtin.

```
my $result = gfx_ray_box_t $x
# or in a pipeline:
@list |> map gfx_ray_box_t |> p
```

gfx_ray_cone_t

gfx_ray_cone_t — batch 47 builtin.

```
my $result = gfx_ray_cone_t $x
# or in a pipeline:
@list |> map gfx_ray_cone_t |> p
```

gfx_ray_cylinder_t

gfx_ray_cylinder_t — batch 47 builtin.

```
my $result = gfx_ray_cylinder_t $x
# or in a pipeline:
@list |> map gfx_ray_cylinder_t |> p
```

gfx_ray_disk_t

gfx_ray_disk_t — batch 47 builtin.

```
my $result = gfx_ray_disk_t $x
# or in a pipeline:
@list |> map gfx_ray_disk_t |> p
```

gfx_ray_ellipsoid_t

gfx_ray_ellipsoid_t — batch 47 builtin.

```
my $result = gfx_ray_ellipsoid_t $x
# or in a pipeline:
@list |> map gfx_ray_ellipsoid_t |> p
```

gfx_ray_plane_t

gfx_ray_plane_t — batch 47 builtin.

```
my $result = gfx_ray_plane_t $x
# or in a pipeline:
@list |> map gfx_ray_plane_t |> p
```

gfx_ray_sphere_t

gfx_ray_sphere_t — batch 47 builtin.

```
my $result = gfx_ray_sphere_t $x
# or in a pipeline:
@list |> map gfx_ray_sphere_t |> p
```

gfx_ray_torus_t_approx

gfx_ray_torus_t_approx — batch 47 builtin.

```
my $result = gfx_ray_torus_t_approx $x
# or in a pipeline:
@list |> map gfx_ray_torus_t_approx |> p
```

gfx_ray_triangle_t

gfx_ray_triangle_t — batch 47 builtin.

```
my $result = gfx_ray_triangle_t $x
# or in a pipeline:
@list |> map gfx_ray_triangle_t |> p
```

gfx_reflect_direction_x

gfx_reflect_direction_x — batch 47 builtin.

```
my $result = gfx_reflect_direction_x $x
# or in a pipeline:
@list |> map gfx_reflect_direction_x |> p
```

gfx_refract_direction_x

gfx_refract_direction_x — batch 47 builtin.

```
my $result = gfx_refract_direction_x $x
# or in a pipeline:
@list |> map gfx_refract_direction_x |> p
```

gfx_rotation_matrix_xx

gfx_rotation_matrix_xx — batch 47 builtin.

```
my $result = gfx_rotation_matrix_xx $x
# or in a pipeline:
@list |> map gfx_rotation_matrix_xx |> p
```

gfx_rotation_matrix_yy

gfx_rotation_matrix_yy — batch 47 builtin.

```
my $result = gfx_rotation_matrix_yy $x
# or in a pipeline:
@list |> map gfx_rotation_matrix_yy |> p
```

gfx_rotation_matrix_zz

gfx_rotation_matrix_zz — batch 47 builtin.

```
my $result = gfx_rotation_matrix_zz $x
# or in a pipeline:
@list |> map gfx_rotation_matrix_zz |> p
```

gfx_scale_matrix_step

gfx_scale_matrix_step — batch 47 builtin.

```
my $result = gfx_scale_matrix_step $x
# or in a pipeline:
@list |> map gfx_scale_matrix_step |> p
```

gfx_screen_space_x

gfx_screen_space_x — batch 47 builtin.

```
my $result = gfx_screen_space_x $x
# or in a pipeline:
@list |> map gfx_screen_space_x |> p
```

gfx_screen_space_y

gfx_screen_space_y — batch 47 builtin.

```
my $result = gfx_screen_space_y $x
# or in a pipeline:
@list |> map gfx_screen_space_y |> p
```

gfx_screen_to_ndc_x

gfx_screen_to_ndc_x — batch 47 builtin.

```
my $result = gfx_screen_to_ndc_x $input
```

gfx_screen_to_ndc_y

gfx_screen_to_ndc_y — batch 47 builtin.

```
my $result = gfx_screen_to_ndc_y $input
```

gfx_shear_matrix_xy

gfx_shear_matrix_xy — batch 47 builtin.

```
my $result = gfx_shear_matrix_xy $x
# or in a pipeline:
@list |> map gfx_shear_matrix_xy |> p
```

gfx_signed_distance_box

gfx_signed_distance_box — batch 47 builtin.

```
my $result = gfx_signed_distance_box $x
# or in a pipeline:
@list |> map gfx_signed_distance_box |> p
```

gfx_signed_distance_capsule

gfx_signed_distance_capsule — batch 47 builtin.

```
my $result = gfx_signed_distance_capsule $x
# or in a pipeline:
@list |> map gfx_signed_distance_capsule |> p
```

gfx_signed_distance_sphere

gfx_signed_distance_sphere — batch 47 builtin.

```
my $result = gfx_signed_distance_sphere $x
# or in a pipeline:
@list |> map gfx_signed_distance_sphere |> p
```

gfx_simplex_noise_step

gfx_simplex_noise_step — batch 47 builtin.

```
my $result = gfx_simplex_noise_step $x
# or in a pipeline:
@list |> map gfx_simplex_noise_step |> p
```

gfx_skybox_uv_u

gfx_skybox_uv_u — batch 47 builtin.

```
my $result = gfx_skybox_uv_u $x
# or in a pipeline:
@list |> map gfx_skybox_uv_u |> p
```

gfx_skybox_uv_v

gfx_skybox_uv_v — batch 47 builtin.

```
my $result = gfx_skybox_uv_v $x
# or in a pipeline:
@list |> map gfx_skybox_uv_v |> p
```

gfx_snells_law_angle

gfx_snells_law_angle — batch 47 builtin.

```
my $result = gfx_snells_law_angle $x
# or in a pipeline:
@list |> map gfx_snells_law_angle |> p
```

gfx_sobol_step

gfx_sobol_step — batch 47 builtin.

```
my $result = gfx_sobol_step $x
# or in a pipeline:
@list |> map gfx_sobol_step |> p
```

gfx_sphere_intersect_t

gfx_sphere_intersect_t — batch 47 builtin.

```
my $result = gfx_sphere_intersect_t $x
# or in a pipeline:
@list |> map gfx_sphere_intersect_t |> p
```

gfx_spherical_harmonic_y00

gfx_spherical_harmonic_y00 — batch 47 builtin.

```
my $result = gfx_spherical_harmonic_y00 $x
# or in a pipeline:
@list |> map gfx_spherical_harmonic_y00 |> p
```

gfx_spherical_harmonic_y10

gfx_spherical_harmonic_y10 — batch 47 builtin.

```
my $result = gfx_spherical_harmonic_y10 $x
# or in a pipeline:
@list |> map gfx_spherical_harmonic_y10 |> p
```

gfx_spherical_harmonic_y11

gfx_spherical_harmonic_y11 — batch 47 builtin.

```
my $result = gfx_spherical_harmonic_y11 $x
# or in a pipeline:
@list |> map gfx_spherical_harmonic_y11 |> p
```

gfx_spherical_harmonic_y20

gfx_spherical_harmonic_y20 — batch 47 builtin.

```
my $result = gfx_spherical_harmonic_y20 $x
# or in a pipeline:
@list |> map gfx_spherical_harmonic_y20 |> p
```

gfx_srgb_to_linear

gfx_srgb_to_linear — batch 47 builtin.

```
my $result = gfx_srgb_to_linear $input
```

gfx_subsurface_scattering_term

gfx_subsurface_scattering_term — batch 47 builtin.

```
my $result = gfx_subsurface_scattering_term $x
# or in a pipeline:
@list |> map gfx_subsurface_scattering_term |> p
```

gfx_sutherland_hodgman

gfx_sutherland_hodgman — batch 47 builtin.

```
my $result = gfx_sutherland_hodgman $x
# or in a pipeline:
@list |> map gfx_sutherland_hodgman |> p
```

gfx_tonemap_aces

gfx_tonemap_aces — batch 47 builtin.

```
my $result = gfx_tonemap_aces $x
# or in a pipeline:
@list |> map gfx_tonemap_aces |> p
```

gfx_tonemap_filmic

gfx_tonemap_filmic — batch 47 builtin.

```
my $result = gfx_tonemap_filmic $x
# or in a pipeline:
@list |> map gfx_tonemap_filmic |> p
```

gfx_tonemap_reinhard

gfx_tonemap_reinhard — batch 47 builtin.

```
my $result = gfx_tonemap_reinhard $x
# or in a pipeline:
@list |> map gfx_tonemap_reinhard |> p
```

gfx_tonemap_uncharted2

gfx_tonemap_uncharted2 — batch 47 builtin.

```
my $result = gfx_tonemap_uncharted2 $x
# or in a pipeline:
@list |> map gfx_tonemap_uncharted2 |> p
```

gfx_total_internal_reflection

gfx_total_internal_reflection — batch 47 builtin.

```
my $result = gfx_total_internal_reflection $x
# or in a pipeline:
@list |> map gfx_total_internal_reflection |> p
```

gfx_translation_matrix_step

gfx_translation_matrix_step — batch 47 builtin.

```
my $result = gfx_translation_matrix_step $x
# or in a pipeline:
@list |> map gfx_translation_matrix_step |> p
```

gfx_translucent_falloff

gfx_translucent_falloff — batch 47 builtin.

```
my $result = gfx_translucent_falloff $x
# or in a pipeline:
@list |> map gfx_translucent_falloff |> p
```

gfx_value_noise_step

gfx_value_noise_step — batch 47 builtin.

```
my $result = gfx_value_noise_step $x
# or in a pipeline:
@list |> map gfx_value_noise_step |> p
```

gfx_van_der_corput

gfx_van_der_corput — batch 47 builtin.

```
my $result = gfx_van_der_corput $x
# or in a pipeline:
@list |> map gfx_van_der_corput |> p
```

gfx_view_matrix_step

`gfx_view_matrix_step` — batch 47 builtin.

```
my $result = gfx_view_matrix_step $x
# or in a pipeline:
@list |> map gfx_view_matrix_step |> p
```

gfx_voronoi_distance

`gfx_voronoi_distance` — batch 47 builtin.

```
my $result = gfx_voronoi_distance $x
# or in a pipeline:
@list |> map gfx_voronoi_distance |> p
```

gfx_worley_noise_step

`gfx_worley_noise_step` — batch 47 builtin.

```
my $result = gfx_worley_noise_step $x
# or in a pipeline:
@list |> map gfx_worley_noise_step |> p
```

gfx_xiaolin_wu_intensity

`gfx_xiaolin_wu_intensity` — batch 47 builtin.

```
my $result = gfx_xiaolin_wu_intensity $x
# or in a pipeline:
@list |> map gfx_xiaolin_wu_intensity |> p
```

gfx_xorshift_step

`gfx_xorshift_step` — batch 47 builtin.

```
my $result = gfx_xorshift_step $x
# or in a pipeline:
@list |> map gfx_xorshift_step |> p
```

gfx_zonal_harmonic_step

`gfx_zonal_harmonic_step` — batch 47 builtin.

```
my $result = gfx_zonal_harmonic_step $x
# or in a pipeline:
@list |> map gfx_zonal_harmonic_step |> p
```

ghz_state

`ghz_state` — batch 80 builtin.

```
my $result = ghz_state $x
# or in a pipeline:
@list |> map ghz_state |> p
```

ghz_state_n

`ghz_state_n` — batches 14-16 builtin.

```
my $result = ghz_state_n $x
# or in a pipeline:
@list |> map ghz_state_n |> p
```

gibbard_satterthwaite_check

`gibbard_satterthwaite_check` — batch 43 builtin.

```
my $result = gibbard_satterthwaite_check $x
# or in a pipeline:
@list |> map gibbard_satterthwaite_check |> p
```

gibbs_thomson_undercooling

`gibbs_thomson_undercooling` — batch 38 builtin.

```
my $result = gibbs_thomson_undercooling $x
# or in a pipeline:
@list |> map gibbs_thomson_undercooling |> p
```

gid

`gid` — system introspection builtin.

```
my $result = gid $x
# or in a pipeline:
@list |> map gid |> p
```

gin

`gin` — algebraic match builtin. Alias for `get_in`.

```
my $result = gin $x
# or in a pipeline:
@list |> map gin |> p
```

gini

`gini(@data)` (alias `gini_coefficient`) — computes the Gini coefficient measuring inequality in a distribution. Returns a value from 0 (perfect equality) to 1 (perfect inequality). Commonly used for income distribution analysis.

```
p gini(1, 1, 1, 1)     # 0 (perfect equality)
p gini(0, 0, 0, 100)   # ~0.75 (high inequality)
```

gini_impurity

`gini_impurity` — batch 27 builtin.

```
my $result = gini_impurity $x
# or in a pipeline:
@list |> map gini_impurity |> p
```

gini_inequality_health

`gini_inequality_health` — batch 61 builtin.

```
my $result = gini_inequality_health $x
# or in a pipeline:
@list |> map gini_inequality_health |> p
```

gini_market

`gini_market` — batch 73 builtin.

```
my $result = gini_market $x
# or in a pipeline:
@list |> map gini_market |> p
```

girth

`girth` — batch 24 builtin.

```
my $result = girth $x
# or in a pipeline:
@list |> map girth |> p
```

girvan_newman

`girvan_newman` — batch 75 builtin.

```
my $result = girvan_newman $x
# or in a pipeline:
@list |> map girvan_newman |> p
```

gk15

`gk15` — batch 19 builtin. Alias for `gauss_kronrod_15`.

```
my $result = gk15 $x
# or in a pipeline:
@list |> map gk15 |> p
```

gl5

`gl5` — batch 19 builtin. Alias for `gauss_legendre_5`.

```
my $result = gl5 $x
# or in a pipeline:
@list |> map gl5 |> p
```

glaisher

`glaisher` — math constants builtin. Alias for `glaisher_constant`.

```
my $result = glaisher $x
# or in a pipeline:
@list |> map glaisher |> p
```

glaisher_constant

`glaisher_constant` — math constants builtin.

```
my $result = glaisher_constant $x
# or in a pipeline:
@list |> map glaisher_constant |> p
```

glicko_rating

`glicko_rating` — batch 56 builtin.

```
my $result = glicko_rating $x
# or in a pipeline:
@list |> map glicko_rating |> p
```

glm

`glm` — batch 53 builtin.

```
my $result = glm $x
# or in a pipeline:
@list |> map glm |> p
```

glob_to_regex

`glob_to_regex` — more regex builtin.

```
my $result = glob_to_regex $input
```

global_efficiency

`global_efficiency` — batches 14-16 builtin.

```
my $result = global_efficiency $x
# or in a pipeline:
@list |> map global_efficiency |> p
```

glushkov_construction

`glushkov_construction` — batch 68 builtin.

```
my $result = glushkov_construction $x
# or in a pipeline:
@list |> map glushkov_construction |> p
```

gmm_moment_condition

`gmm_moment_condition` — batch 36 builtin.

```
my $result = gmm_moment_condition $x
# or in a pipeline:
@list |> map gmm_moment_condition |> p
```

gmm_moment_function

`gmm_moment_function` — batch 36 builtin.

```
my $result = gmm_moment_function $x
# or in a pipeline:
@list |> map gmm_moment_function |> p
```

go_enrichment_p

`go_enrichment_p` — batch 33 builtin.

```
my $result = go_enrichment_p $x
# or in a pipeline:
@list |> map go_enrichment_p |> p
```

goals_above_avg

`goals_above_avg` — batch 56 builtin.

```
my $result = goals_above_avg $x
# or in a pipeline:
@list |> map goals_above_avg |> p
```

goertzel

`goertzel` computes the magnitude of a single DFT frequency bin using the Goertzel algorithm. Much faster than full FFT when you need one frequency.

```
my $mag = goertzel(\@signal, 440, 44100)  # 440 Hz component
```

goldb

`goldb` — math / numeric (uncategorized batch) builtin. Alias for `goldbach`.

```
my $result = goldb $x
# or in a pipeline:
@list |> map goldb |> p
```

goldbach

`goldbach` — math / numeric (uncategorized batch) builtin.

```
my $result = goldbach $x
# or in a pipeline:
@list |> map goldbach |> p
```

goldbach_pair

`goldbach_pair` — batch 35 builtin.

```
my $result = goldbach_pair $x
# or in a pipeline:
@list |> map goldbach_pair |> p
```

golden

`golden_section` (aliases `golden`, `gss`) finds the minimum of f on [a,b] via golden-section search.

```
my $xmin = golden(fn { ($_[0]-3)**2 }, 0, 10)  # 3.0
```

golden_ratio

`golden_ratio` (alias `phi`) — φ = (1+√5)/2 ≈ 1.618. The golden ratio, appears throughout nature and art.

```
p phi   # 1.6180339887498895
# Fibonacci limit: fib(n)/fib(n-1) → φ
```

goldfeld_quandt

`goldfeld_quandt` — batch 36 builtin.

```
my $result = goldfeld_quandt $x
# or in a pipeline:
@list |> map goldfeld_quandt |> p
```

golomb_rice_code

`golomb_rice_code` — batch 40 builtin.

```
my $result = golomb_rice_code $x
# or in a pipeline:
@list |> map golomb_rice_code |> p
```

golstep

`golstep` — algorithms / puzzles builtin. Alias for `game_of_life_step`.

```
my $result = golstep $x
# or in a pipeline:
@list |> map golstep |> p
```

gomory_cut_step

`gomory_cut_step` — batch 41 builtin.

```
my $result = gomory_cut_step $x
# or in a pipeline:
@list |> map gomory_cut_step |> p
```

gomory_hu_step

`gomory_hu_step` — batch 41 builtin.

```
my $result = gomory_hu_step $x
# or in a pipeline:
@list |> map gomory_hu_step |> p
```

gompertz_growth_step

`gompertz_growth_step` — batch 22 builtin.

```
my $result = gompertz_growth_step $x
# or in a pipeline:
@list |> map gompertz_growth_step |> p
```

gompertz_pdf

`gompertz_pdf` — batches 17-18 builtin.

```
my $result = gompertz_pdf $x
# or in a pipeline:
@list |> map gompertz_pdf |> p
```

goto

`goto` — control flow builtin.

```
my $result = goto $x
# or in a pipeline:
@list |> map goto |> p
```

gouy_chapman_potential

`gouy_chapman_potential` — batch 38 builtin.

```
my $result = gouy_chapman_potential $x
# or in a pipeline:
@list |> map gouy_chapman_potential |> p
```

gradient_descent_step

`gradient_descent_step` — batch 19 builtin.

```
my $result = gradient_descent_step $x
# or in a pipeline:
@list |> map gradient_descent_step |> p
```

gradient_geothermal

`gradient_geothermal` — batch 65 builtin.

```
my $result = gradient_geothermal $x
# or in a pipeline:
@list |> map gradient_geothermal |> p
```

gradient_richardson_full

`gradient_richardson_full` — batch 42 builtin.

```
my $result = gradient_richardson_full $x
# or in a pipeline:
@list |> map gradient_richardson_full |> p
```

gradient_temporal_diff

`gradient_temporal_diff` — batch 40 builtin.

```
my $result = gradient_temporal_diff $x
# or in a pipeline:
@list |> map gradient_temporal_diff |> p
```

gradient_wind_step

`gradient_wind_step` — batch 42 builtin.

```
my $result = gradient_wind_step $x
# or in a pipeline:
@list |> map gradient_wind_step |> p
```

graham_2approx_bound

`graham_2approx_bound` — batch 41 builtin.

```
my $result = graham_2approx_bound $x
# or in a pipeline:
@list |> map graham_2approx_bound |> p
```

gram_schmidt

`gram_schmidt` — batch 11 builtin. Alias for `orthogonalize_vectors`.

```
my $result = gram_schmidt $x
# or in a pipeline:
@list |> map gram_schmidt |> p
```

granger_causality

`granger_causality` — batch 78 builtin.

```
my $result = granger_causality $x
# or in a pipeline:
@list |> map granger_causality |> p
```

granger_causality_chi2

granger_causality_chi2 — batch 36 builtin.

```
my $result = granger_causality_chi2 $x
# or in a pipeline:
@list |> map granger_causality_chi2 |> p
```

graph_average_degree

graph_average_degree — batch 13 builtin.

```
my $result = graph_average_degree $x
# or in a pipeline:
@list |> map graph_average_degree |> p
```

graph_avg_clustering

graph_avg_clustering — batches 17-18 builtin.

```
my $result = graph_avg_clustering $x
# or in a pipeline:
@list |> map graph_avg_clustering |> p
```

graph_bipartite_match_simple

graph_bipartite_match_simple — batches 17-18 builtin.

```
my $result = graph_bipartite_match_simple $x
# or in a pipeline:
@list |> map graph_bipartite_match_simple |> p
```

graph_coloring_brooks_bound

graph_coloring_brooks_bound — batch 41 builtin.

```
my $result = graph_coloring_brooks_bound $x
# or in a pipeline:
@list |> map graph_coloring_brooks_bound |> p
```

graph_coloring_dsatur_step

graph_coloring_dsatur_step — batch 41 builtin.

```
my $result = graph_coloring_dsatur_step $x
# or in a pipeline:
@list |> map graph_coloring_dsatur_step |> p
```

graph_coloring_greedy

graph_coloring_greedy — batch 75 builtin.

```
my $result = graph_coloring_greedy $x
# or in a pipeline:
@list |> map graph_coloring_greedy |> p
```

graph_coloring_lp_bound

graph_coloring_lp_bound — batch 41 builtin.

```
my $result = graph_coloring_lp_bound $x
# or in a pipeline:
@list |> map graph_coloring_lp_bound |> p
```

graph_coloring_welsh_powell

graph_coloring_welsh_powell — batch 41 builtin.

```
my $result = graph_coloring_welsh_powell $x
# or in a pipeline:
@list |> map graph_coloring_welsh_powell |> p
```

graph_complement

graph_complement — batch 13 builtin.

```
my $result = graph_complement $x
# or in a pipeline:
@list |> map graph_complement |> p
```

graph_count_edges

graph_count_edges — batches 17-18 builtin.

```
my $result = graph_count_edges $x
# or in a pipeline:
@list |> map graph_count_edges |> p
```

graph_count_paths_length_k

graph_count_paths_length_k — batches 17-18 builtin.

```
my $result = graph_count_paths_length_k $x
# or in a pipeline:
@list |> map graph_count_paths_length_k |> p
```

graph_count_triangles

graph_count_triangles — batches 17-18 builtin.

```
my $result = graph_count_triangles $x
# or in a pipeline:
@list |> map graph_count_triangles |> p
```

graph_cut_segment

graph_cut_segment — batch 77 builtin.

```
my $result = graph_cut_segment $x
# or in a pipeline:
@list |> map graph_cut_segment |> p
```

graph_degree_distribution

graph_degree_distribution — batches 17-18 builtin.

```
my $result = graph_degree_distribution $x
# or in a pipeline:
@list |> map graph_degree_distribution |> p
```

graph_density

graph_density — batch 13 builtin.

```
my $result = graph_density $x
# or in a pipeline:
@list |> map graph_density |> p
```

graph_eccentricity_all

`graph_eccentricity_all` — batch 13 builtin.

```
my $result = graph_eccentricity_all $x
# or in a pipeline:
@list |> map graph_eccentricity_all |> p
```

graph_independent_set_brute

`graph_independent_set_brute` — batches 17-18 builtin.

```
my $result = graph_independent_set_brute $x
# or in a pipeline:
@list |> map graph_independent_set_brute |> p
```

graph_is_tree

`graph_is_tree` — batch 13 builtin.

```
my $result = graph_is_tree $x
# or in a pipeline:
@list |> map graph_is_tree |> p
```

graph_max_clique_brute

`graph_max_clique_brute` — batches 17-18 builtin.

```
my $result = graph_max_clique_brute $x
# or in a pipeline:
@list |> map graph_max_clique_brute |> p
```

graph_max_degree

`graph_max_degree` — batch 13 builtin.

```
my $result = graph_max_degree $x
# or in a pipeline:
@list |> map graph_max_degree |> p
```

graph_min_degree

`graph_min_degree` — batch 13 builtin.

```
my $result = graph_min_degree $x
# or in a pipeline:
@list |> map graph_min_degree |> p
```

graph_pagerank_simple

`graph_pagerank_simple` — batches 17-18 builtin.

```
my $result = graph_pagerank_simple $x
# or in a pipeline:
@list |> map graph_pagerank_simple |> p
```

graph_transitivity

`graph_transitivity` — batches 17-18 builtin.

```
my $result = graph_transitivity $x
# or in a pipeline:
@list |> map graph_transitivity |> p
```

grashof_number

`grashof_number` — batch 13 builtin.

```
my $result = grashof_number $x
# or in a pipeline:
@list |> map grashof_number |> p
```

grashof_number_step

`grashof_number_step` — batch 42 builtin.

```
my $result = grashof_number_step $x
# or in a pipeline:
@list |> map grashof_number_step |> p
```

grav_binding_energy

`grav_binding_energy` — batch 31 builtin.

```
my $result = grav_binding_energy $x
# or in a pipeline:
@list |> map grav_binding_energy |> p
```

grav_dilation_factor

`grav_dilation_factor` — batch 31 builtin.

```
my $result = grav_dilation_factor $x
# or in a pipeline:
@list |> map grav_dilation_factor |> p
```

grav_redshift

`grav_redshift` — batch 23 builtin.

```
my $result = grav_redshift $x
# or in a pipeline:
@list |> map grav_redshift |> p
```

grav_time_dilation

`grav_time_dilation` — batch 23 builtin.

```
my $result = grav_time_dilation $x
# or in a pipeline:
@list |> map grav_time_dilation |> p
```

gravitational_constant

`gravitational_constant` — constants builtin.

```
my $result = gravitational_constant $x
# or in a pipeline:
@list |> map gravitational_constant |> p
```

gravitational_pe

`gravitational_pe` — batch 12 builtin.

```
my $result = gravitational_pe $x
# or in a pipeline:
@list |> map gravitational_pe |> p
```

gravitational_wave_quadrupole

`gravitational_wave_quadrupole` — batch 39 builtin.

```
my $result = gravitational_wave_quadrupole $x
# or in a pipeline:
@list |> map gravitational_wave_quadrupole |> p
```

gravity

`gravity` — math / physics constants builtin.

```
my $result = gravity $x
# or in a pipeline:
@list |> map gravity |> p
```

gravity_at_radius

`gravity_at_radius` — batch 12 builtin.

```
my $result = gravity_at_radius $x
# or in a pipeline:
@list |> map gravity_at_radius |> p
```

gray

`gray` — color / ansi builtin. Alias for `red`.

```
my $result = gray $x
# or in a pipeline:
@list |> map gray |> p
```

gray2b

`gray2b` — base / gray code builtin. Alias for `gray_to_binary`.

```
my $result = gray2b $x
# or in a pipeline:
@list |> map gray2b |> p
```

gray_code_sequence

`gray_code_sequence` — base / gray code builtin.

```
my $result = gray_code_sequence $x
# or in a pipeline:
@list |> map gray_code_sequence |> p
```

gray_to_bin

`gray_to_bin` — batch 10 builtin.

```
my $result = gray_to_bin $input
```

gray_to_binary

`gray_to_binary` — base / gray code builtin.

```
my $result = gray_to_binary $input
```

grayseq

`grayseq` — base / gray code builtin. Alias for `gray_code_sequence`.

```
my $result = grayseq $x
# or in a pipeline:
@list |> map grayseq |> p
```

great_circle_law_of_cos

`great_circle_law_of_cos` — batch 28 builtin.

```
my $result = great_circle_law_of_cos $x
# or in a pipeline:
@list |> map great_circle_law_of_cos |> p
```

greedy_coloring

`greedy_coloring` — batch 24 builtin.

```
my $result = greedy_coloring $x
# or in a pipeline:
@list |> map greedy_coloring |> p
```

greedy_edge_tour

`greedy_edge_tour` — batch 41 builtin.

```
my $result = greedy_edge_tour $x
# or in a pipeline:
@list |> map greedy_edge_tour |> p
```

greedy_set_cover_round

`greedy_set_cover_round` — batch 41 builtin.

```
my $result = greedy_set_cover_round $x
# or in a pipeline:
@list |> map greedy_set_cover_round |> p
```

green

`green` — color / ansi builtin. Alias for `red`.

```
my $result = green $x
# or in a pipeline:
@list |> map green |> p
```

green_bold

`green_bold` — color / ansi builtin. Alias for `red`.

```
my $result = green_bold $x
# or in a pipeline:
@list |> map green_bold |> p
```

greenhouse_forcing_step

`greenhouse_forcing_step` — batch 42 builtin.

```
my $result = greenhouse_forcing_step $x
# or in a pipeline:
@list |> map greenhouse_forcing_step |> p
```

gregorian_from_fixed

`gregorian_from_fixed` — batch 52 builtin.

```
my $result = gregorian_from_fixed $x
# or in a pipeline:
@list |> map gregorian_from_fixed |> p
```

grepv

`grepv` — functional / iterator builtin. Alias for `take_while`.

```
my $result = grepv $x
# or in a pipeline:
@list |> map grepv |> p
```

grey

`grey` — color / ansi builtin. Alias for `red`.

```
my $result = grey $x
# or in a pipeline:
@list |> map grey |> p
```

grim_trigger_step

`grim_trigger_step` — batch 43 builtin.

```
my $result = grim_trigger_step $x
# or in a pipeline:
@list |> map grim_trigger_step |> p
```

gromov_wasserstein_step

`gromov_wasserstein_step` — batch 40 builtin.

```
my $result = gromov_wasserstein_step $x
# or in a pipeline:
@list |> map gromov_wasserstein_step |> p
```

gromov_witten_invariant

`gromov_witten_invariant` — batch 37 builtin.

```
my $result = gromov_witten_invariant $x
# or in a pipeline:
@list |> map gromov_witten_invariant |> p
```

gross_premium_load

`gross_premium_load` — batch 60 builtin.

```
my $result = gross_premium_load $x
# or in a pipeline:
@list |> map gross_premium_load |> p
```

group_by

`group_by` — functional / iterator builtin.

```
my $result = group_by $x
# or in a pipeline:
@list |> map group_by |> p
```

group_by_fn

`group_by_fn` — go/general functional utilities builtin.

```
my $result = group_by_fn $x
# or in a pipeline:
@list |> map group_by_fn |> p
```

group_by_size

`group_by_size` — list helpers (batch 4) builtin.

```
my $result = group_by_size $x
# or in a pipeline:
@list |> map group_by_size |> p
```

group_cohomology_dim

`group_cohomology_dim` — batch 37 builtin.

```
my $result = group_cohomology_dim $x
# or in a pipeline:
@list |> map group_cohomology_dim |> p
```

group_consecutive

`group_consecutive` — functional combinators builtin.

```
my $result = group_consecutive $x
# or in a pipeline:
@list |> map group_consecutive |> p
```

group_consecutive_by

`group_consecutive_by` — matrix operations (uncategorized batch) builtin.

```
my $result = group_consecutive_by $x
# or in a pipeline:
@list |> map group_consecutive_by |> p
```

group_delay_eval

`group_delay_eval` — batch 74 builtin.

```
my $result = group_delay_eval $x
# or in a pipeline:
@list |> map group_delay_eval |> p
```

group_homology_dim

`group_homology_dim` — batch 37 builtin.

```
my $result = group_homology_dim $x
# or in a pipeline:
@list |> map group_homology_dim |> p
```

group_number

`group_number` — file stat / path builtin. Alias for `format_number`.

```
my $result = group_number $x
# or in a pipeline:
@list |> map group_number |> p
```

group_of_n

`group_of_n` — collection (batch 2) builtin.

```
my $result = group_of_n $x
# or in a pipeline:
@list |> map group_of_n |> p
```

group_runs

`group_runs` — haskell list functions builtin.

```
my $result = group_runs $x
# or in a pipeline:
@list |> map group_runs |> p
```

groupby_iter

`groupby_iter` — python/ruby stdlib builtin.

```
my $result = groupby_iter $x
# or in a pipeline:
@list |> map groupby_iter |> p
```

grover_iter

`grover_iter` — batch 80 builtin.

```
my $result = grover_iter $x
# or in a pipeline:
@list |> map grover_iter |> p
```

growing_perpetuity

`growing_perpetuity` — batch 13 builtin.

```
my $result = growing_perpetuity $x
# or in a pipeline:
@list |> map growing_perpetuity |> p
```

growth_rate_from_ratio

`growth_rate_from_ratio` — batch 22 builtin.

```
my $result = growth_rate_from_ratio $x
# or in a pipeline:
@list |> map growth_rate_from_ratio |> p
```

gruns

`gruns` — haskell list functions builtin. Alias for `group_runs`.

```
my $result = gruns $x
# or in a pipeline:
@list |> map gruns |> p
```

gullstrand_painleve_step

`gullstrand_painleve_step` — batch 39 builtin.

```
my $result = gullstrand_painleve_step $x
# or in a pipeline:
@list |> map gullstrand_painleve_step |> p
```

gunning_fog

`gunning_fog` — batch 9 builtin.

```
my $result = gunning_fog $x
# or in a pipeline:
@list |> map gunning_fog |> p
```

gutenberg_richter_b

`gutenberg_richter_b` — batch 65 builtin.

```
my $result = gutenberg_richter_b $x
# or in a pipeline:
@list |> map gutenberg_richter_b |> p
```

gw_strain_amplitude

`gw_strain_amplitude` — batch 31 builtin.

```
my $result = gw_strain_amplitude $x
# or in a pipeline:
@list |> map gw_strain_amplitude |> p
```

gx_diameter

`gx_diameter` — batch 75 builtin.

```
my $result = gx_diameter $x
# or in a pipeline:
@list |> map gx_diameter |> p
```

gx_eccentricity

`gx_eccentricity` — batch 75 builtin.

```
my $result = gx_eccentricity $x
# or in a pipeline:
@list |> map gx_eccentricity |> p
```

gx_radius

`gx_radius` — batch 75 builtin.

```
my $result = gx_radius $x
# or in a pipeline:
@list |> map gx_radius |> p
```

gz

`gz` — crypto / encoding builtin. Alias for `gzip`.

```
my $result = gz $x
# or in a pipeline:
@list |> map gz |> p
```

gzip_crc32_init

`gzip_crc32_init` — batch 62 builtin.

```
my $result = gzip_crc32_init $x
# or in a pipeline:
@list |> map gzip_crc32_init |> p
```

gzip_encode_step

`gzip_encode_step` — batch 81 builtin.

```
my $result = gzip_encode_step $x
# or in a pipeline:
@list |> map gzip_encode_step |> p
```

gzip_isize

`gzip_isize` — batch 62 builtin.

```
my $result = gzip_isize $x
# or in a pipeline:
@list |> map gzip_isize |> p
```

gzip_member_step

`gzip_member_step` — batch 62 builtin.

```
my $result = gzip_member_step $x
# or in a pipeline:
@list |> map gzip_member_step |> p
```

h0

`h0` — physics constants builtin. Alias for `hubble_constant`.

```
my $result = h0 $x
# or in a pipeline:
@list |> map h0 |> p
```

h3_geo_to_h3

`h3_geo_to_h3` — batch 58 builtin.

```
my $result = h3_geo_to_h3 $input
```

h3_h3_to_geo

`h3_h3_to_geo` — batch 58 builtin.

```
my $result = h3_h3_to_geo $input
```

h3_index

`h3_index` — batch 58 builtin.

```
my $result = h3_index $x
# or in a pipeline:
@list |> map h3_index |> p
```

h3_k_ring

`h3_k_ring` — batch 58 builtin.

```
my $result = h3_k_ring $x
# or in a pipeline:
@list |> map h3_k_ring |> p
```

h3_neighbor

`h3_neighbor` — batch 58 builtin.

```
my $result = h3_neighbor $x
# or in a pipeline:
@list |> map h3_neighbor |> p
```

h3_resolution

`h3_resolution` — batch 58 builtin.

```
my $result = h3_resolution $x
# or in a pipeline:
@list |> map h3_resolution |> p
```

h_infinity_norm

`h_infinity_norm` — batch 59 builtin.

```
my $result = h_infinity_norm $x
# or in a pipeline:
@list |> map h_infinity_norm |> p
```

haar_2d_step

`haar_2d_step` — batches 14-16 builtin.

```
my $result = haar_2d_step $x
# or in a pipeline:
@list |> map haar_2d_step |> p
```

hadamard_walsh_transform_step

`hadamard_walsh_transform_step` — batch 40 builtin.

```
my $result = hadamard_walsh_transform_step $x
# or in a pipeline:
@list |> map hadamard_walsh_transform_step |> p
```

hadley_cell_max_lat

`hadley_cell_max_lat` — batch 42 builtin.

```
my $result = hadley_cell_max_lat $x
# or in a pipeline:
@list |> map hadley_cell_max_lat |> p
```

hagen_poiseuille_eo

`hagen_poiseuille_eo` — batch 38 builtin.

```
my $result = hagen_poiseuille_eo $x
# or in a pipeline:
@list |> map hagen_poiseuille_eo |> p
```

half

`half` — trivial numeric / predicate builtins builtin.

```
my $result = half $x
# or in a pipeline:
@list |> map half |> p
```

half_cauchy_pdf

`half_cauchy_pdf` — batches 17-18 builtin.

```
my $result = half_cauchy_pdf $x
# or in a pipeline:
@list |> map half_cauchy_pdf |> p
```

half_each

`half_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = half_each $x
# or in a pipeline:
@list |> map half_each |> p
```

half_logistic_pdf

`half_logistic_pdf` — batches 17-18 builtin.

```
my $result = half_logistic_pdf $x
# or in a pipeline:
@list |> map half_logistic_pdf |> p
```

half_reaction_balance

`half_reaction_balance` — batch 63 builtin.

```
my $result = half_reaction_balance $x
# or in a pipeline:
@list |> map half_reaction_balance |> p
```

hall_inner_product_two

`hall_inner_product_two` — batch 37 builtin.

```
my $result = hall_inner_product_two $x
# or in a pipeline:
@list |> map hall_inner_product_two |> p
```

halley

`halley` — batch 19 builtin. Alias for `halley_root`.

```
my $result = halley $x
# or in a pipeline:
@list |> map halley |> p
```

halley_root

`halley_root` — batch 19 builtin.

```
my $result = halley_root $x
# or in a pipeline:
@list |> map halley_root |> p
```

hamdist

`hamdist` — extended stdlib batch 3 builtin.

```
my $result = hamdist $x
# or in a pipeline:
@list |> map hamdist |> p
```

hamdist_int

`hamdist_int` — batch 10 builtin. Alias for `hamming_distance_int`.

```
my $result = hamdist_int $x
# or in a pipeline:
@list |> map hamdist_int |> p
```

hamiltonian_brute

`hamiltonian_brute` — batch 13 builtin.

```
my $result = hamiltonian_brute $x
# or in a pipeline:
@list |> map hamiltonian_brute |> p
```

hamiltonian_path

`hamiltonian_path` — batch 75 builtin.

```
my $result = hamiltonian_path $x
# or in a pipeline:
@list |> map hamiltonian_path |> p
```

hamming

`window_hamming($n)` (alias `hamming`) — generates a Hamming window of length n. Similar to Hann but with slightly different coefficients, optimized for speech processing.

```
my $w = hamming(1024)
```

hamming_distance

`hamming_distance` — string (batch 2) builtin.

```
my $result = hamming_distance $x
# or in a pipeline:
@list |> map hamming_distance |> p
```

hamming_distance_int

`hamming_distance_int` — batch 10 builtin.

```
my $result = hamming_distance_int $x
# or in a pipeline:
@list |> map hamming_distance_int |> p
```

hamming_distance_str

`hamming_distance_str` — batch 28 builtin.

```
my $result = hamming_distance_str $x
# or in a pipeline:
@list |> map hamming_distance_str |> p
```

hamming_protein

`hamming_protein` — batch 33 builtin.

```
my $result = hamming_protein $x
# or in a pipeline:
@list |> map hamming_protein |> p
```

hamming_w

`hamming_w` — batch 74 builtin.

```
my $result = hamming_w $x
# or in a pipeline:
@list |> map hamming_w |> p
```

hamming_weight

`hamming_weight` — batch 10 builtin.

```
my $result = hamming_weight $x
# or in a pipeline:
@list |> map hamming_weight |> p
```

hamming_window

`hamming_window` — batch 25 builtin.

```
my $result = hamming_window $x
# or in a pipeline:
@list |> map hamming_window |> p
```

hann

`window_hann($n)` (alias `hann`) — generates a Hann (raised cosine) window of length n. Common window function for spectral analysis that reduces spectral leakage.

```
my $w = hann(1024)
```

hann_w

`hann_w` — batch 74 builtin.

```
my $result = hann_w $x
# or in a pipeline:
@list |> map hann_w |> p
```

hann_window

`hann_window` — batch 25 builtin.

```
my $result = hann_window $x
# or in a pipeline:
@list |> map hann_window |> p
```

hannan_quinn_ic

`hannan_quinn_ic` — batch 36 builtin.

```
my $result = hannan_quinn_ic $x
# or in a pipeline:
@list |> map hannan_quinn_ic |> p
```

hanoi

`hanoi` — algorithms / puzzles builtin. Alias for `tower_of_hanoi`.

```
my $result = hanoi $x
# or in a pipeline:
@list |> map hanoi |> p
```

hansen_j_test

`hansen_j_test` — batch 36 builtin.

```
my $result = hansen_j_test $x
# or in a pipeline:
@list |> map hansen_j_test |> p
```

hard_sigmoid

`hard_sigmoid` — ml activation functions builtin.

```
my $result = hard_sigmoid $x
# or in a pipeline:
@list |> map hard_sigmoid |> p
```

hard_swish

`hard_swish` — ml activation functions builtin.

```
my $result = hard_swish $x
# or in a pipeline:
@list |> map hard_swish |> p
```

hardsigmoid

`hardsigmoid` — ml activation functions builtin. Alias for `hard_sigmoid`.

```
my $result = hardsigmoid $x
# or in a pipeline:
@list |> map hardsigmoid |> p
```

hardswish

`hardswish` — ml activation functions builtin. Alias for `hard_swish`.

```
my $result = hardswish $x
# or in a pipeline:
@list |> map hardswish |> p
```

harmonic

`harmonic_number` (alias `harmonic`) computes the nth harmonic number H_n = 1 + 1/2 + 1/3 + ... + 1/n.

```
p harmonic(1)    # 1.0
p harmonic(10)   # 2.9290
p harmonic(100)  # 5.1874
```

harmonic_centrality

`harmonic_centrality` — batch 75 builtin.

```
my $result = harmonic_centrality $x
# or in a pipeline:
@list |> map harmonic_centrality |> p
```

harmonic_mean

`harmonic_mean` — file stat / path builtin.

```
my $result = harmonic_mean $x
# or in a pipeline:
@list |> map harmonic_mean |> p
```

harmonic_mean_arr

`harmonic_mean_arr` — batch 11 builtin.

```
my $result = harmonic_mean_arr $x
# or in a pipeline:
@list |> map harmonic_mean_arr |> p
```

harmonic_oscillator_energy

`harmonic_oscillator_energy` — batch 32 builtin.

```
my $result = harmonic_oscillator_energy $x
# or in a pipeline:
@list |> map harmonic_oscillator_energy |> p
```

harmonic_seq_sum

`harmonic_seq_sum` — batches 14-16 builtin.

```
my $result = harmonic_seq_sum $x
# or in a pipeline:
@list |> map harmonic_seq_sum |> p
```

harmonics_partial

`harmonics_partial` — batch 64 builtin.

```
my $result = harmonics_partial $x
# or in a pipeline:
@list |> map harmonics_partial |> p
```

harris_corners

harris_corners — batch 77 builtin.

```
my $result = harris_corners $x
# or in a pipeline:
@list |> map harris_corners |> p
```

hartle_thorne_metric

hartle_thorne_metric — batch 39 builtin.

```
my $result = hartle_thorne_metric $x
# or in a pipeline:
@list |> map hartle_thorne_metric |> p
```

has_all_keys

has_all_keys — hash helpers builtin.

```
my $result = has_all_keys $x
# or in a pipeline:
@list |> map has_all_keys |> p
```

has_any_key

has_any_key — hash helpers builtin.

```
my $result = has_any_key $x
# or in a pipeline:
@list |> map has_any_key |> p
```

has_cycle_directed

has_cycle_directed — batch 24 builtin.

```
my $result = has_cycle_directed $x
# or in a pipeline:
@list |> map has_cycle_directed |> p
```

has_cycle_graph

has_cycle_graph — extended stdlib builtin.

```
my $result = has_cycle_graph $x
# or in a pipeline:
@list |> map has_cycle_graph |> p
```

has_cycle_undirected

has_cycle_undirected — batch 24 builtin.

```
my $result = has_cycle_undirected $x
# or in a pipeline:
@list |> map has_cycle_undirected |> p
```

has_duplicates

has_duplicates — more list helpers builtin.

```
my $result = has_duplicates $x
# or in a pipeline:
@list |> map has_duplicates |> p
```

has_key

`has_key` — hash helpers builtin.

```
my $result = has_key $x
# or in a pipeline:
@list |> map has_key |> p
```

has_stderr_tty

`has_stderr_tty` — process / env builtin.

```
my $result = has_stderr_tty $x
# or in a pipeline:
@list |> map has_stderr_tty |> p
```

has_stdin_tty

`has_stdin_tty` — process / env builtin.

```
my $result = has_stdin_tty $x
# or in a pipeline:
@list |> map has_stdin_tty |> p
```

has_stdout_tty

`has_stdout_tty` — process / env builtin.

```
my $result = has_stdout_tty $x
# or in a pipeline:
@list |> map has_stdout_tty |> p
```

hascyc

`hascyc` — extended stdlib builtin. Alias for `has_cycle_graph`.

```
my $result = hascyc $x
# or in a pipeline:
@list |> map hascyc |> p
```

hash_combine

`hash_combine` — batch 35 builtin.

```
my $result = hash_combine $x
# or in a pipeline:
@list |> map hash_combine |> p
```

hash_delete

`hash_delete` — file stat / path builtin.

```
my $result = hash_delete $x
# or in a pipeline:
@list |> map hash_delete |> p
```

hash_eq

`hash_eq` — hash ops (batch 2) builtin.

```
my $result = hash_eq $x
# or in a pipeline:
@list |> map hash_eq |> p
```

hash_filter_keys

`hash_filter_keys` — hash helpers builtin.

```
my $result = hash_filter_keys $x
# or in a pipeline:
@list |> map hash_filter_keys |> p
```

hash_filter_values

`hash_filter_values` — hash helpers builtin.

```
my $result = hash_filter_values $x
# or in a pipeline:
@list |> map hash_filter_values |> p
```

hash_from_list

`hash_from_list` — list helpers (batch 4) builtin.

```
my $result = hash_from_list $x
# or in a pipeline:
@list |> map hash_from_list |> p
```

hash_from_pairs

`hash_from_pairs` — hash ops (batch 2) builtin.

```
my $result = hash_from_pairs $x
# or in a pipeline:
@list |> map hash_from_pairs |> p
```

hash_insert

`hash_insert` — file stat / path builtin.

```
my $result = hash_insert $x
# or in a pipeline:
@list |> map hash_insert |> p
```

hash_map_values

`hash_map_values` — hash helpers builtin.

```
my $result = hash_map_values $x
# or in a pipeline:
@list |> map hash_map_values |> p
```

hash_merge_deep

`hash_merge_deep` — list helpers (batch 4) builtin.

```
my $result = hash_merge_deep $x
# or in a pipeline:
@list |> map hash_merge_deep |> p
```

hash_size

`hash_size` — hash ops (batch 2) builtin.

```
my $result = hash_size $x
# or in a pipeline:
@list |> map hash_size |> p
```

hash_to_list

`hash_to_list` — list helpers (batch 4) builtin.

```
my $result = hash_to_list $input
```

hash_update

`hash_update` — file stat / path builtin.

```
my $result = hash_update $x
# or in a pipeline:
@list |> map hash_update |> p
```

hash_zip

`hash_zip` — list helpers (batch 4) builtin.

```
my $result = hash_zip $x
# or in a pipeline:
@list |> map hash_zip |> p
```

hausdorff_distance_2d

`hausdorff_distance_2d` — batches 17-18 builtin.

```
my $result = hausdorff_distance_2d $x
# or in a pipeline:
@list |> map hausdorff_distance_2d |> p
```

hausman_test

`hausman_test` — batch 36 builtin.

```
my $result = hausman_test $x
# or in a pipeline:
@list |> map hausman_test |> p
```

havdist

`havdist` — geometry (extended) builtin. Alias for `haversine_distance`.

```
my $result = havdist $x
# or in a pipeline:
@list |> map havdist |> p
```

haversine

`haversine_distance($lat1, $lon1, $lat2, $lon2)` (alias `haversine`) — computes the great-circle distance between two points on Earth given latitude/longitude in degrees. Returns distance in kilometers.

```
# NYC to LA
p haversine(40.7128, -74.0060, 34.0522, -118.2437)
# ~3940 km
```

haversine_dist

`haversine_dist` — batch 58 builtin.

```
my $result = haversine_dist $x
# or in a pipeline:
@list |> map haversine_dist |> p
```

hawk_dove_payoff

`hawk_dove_payoff` — batch 43 builtin.

```
my $result = hawk_dove_payoff $x
# or in a pipeline:
@list |> map hawk_dove_payoff |> p
```

hawking_area_increase

`hawking_area_increase` — batch 39 builtin.

```
my $result = hawking_area_increase $x
# or in a pipeline:
@list |> map hawking_area_increase |> p
```

hclust

`hclust` — hierarchical clustering (average linkage). Takes a distance matrix. Returns merge list [[i,j,height],...]. Like R's hclust().

```
my $merges = hclust(dist_mat([[0,0],[1,0],[10,10]]))
```

head_finding_collins

`head_finding_collins` — batch 69 builtin.

```
my $result = head_finding_collins $x
# or in a pipeline:
@list |> map head_finding_collins |> p
```

head_lines

`head_lines` — file stat / path builtin.

```
my $result = head_lines $x
# or in a pipeline:
@list |> map head_lines |> p
```

head_n

`head_n` — list helpers (batch 4) builtin.

```
my $result = head_n $x
# or in a pipeline:
@list |> map head_n |> p
```

head_str

`head_str` — string (batch 2) builtin. Alias for `left_str`.

```
my $result = head_str $x
# or in a pipeline:
@list |> map head_str |> p
```

heap_sift_down

`heap_sift_down` — batch 10 builtin.

```
my $result = heap_sift_down $x
# or in a pipeline:
@list |> map heap_sift_down |> p
```

heat_capacity_q

`heat_capacity_q` — batch 21 builtin.

```
my $result = heat_capacity_q $x
# or in a pipeline:
@list |> map heat_capacity_q |> p
```

heat_flow_radiogenic

`heat_flow_radiogenic` — batch 65 builtin.

```
my $result = heat_flow_radiogenic $x
# or in a pipeline:
@list |> map heat_flow_radiogenic |> p
```

heat_index

`heat_index` — physics formulas builtin.

```
my $result = heat_index $x
# or in a pipeline:
@list |> map heat_index |> p
```

heat_index_celsius

`heat_index_celsius` — batch 12 builtin.

```
my $result = heat_index_celsius $x
# or in a pipeline:
@list |> map heat_index_celsius |> p
```

hebrew_leap_year

`hebrew_leap_year` — batch 52 builtin.

```
my $result = hebrew_leap_year $x
# or in a pipeline:
@list |> map hebrew_leap_year |> p
```

hebrew_year_length

`hebrew_year_length` — batch 52 builtin.

```
my $result = hebrew_year_length $x
# or in a pipeline:
@list |> map hebrew_year_length |> p
```

heckman_correction

heckman_correction — batch 36 builtin.

```
my $result = heckman_correction $x
# or in a pipeline:
@list |> map heckman_correction |> p
```

heegaard_genus_lower

heegaard_genus_lower — batch 37 builtin.

```
my $result = heegaard_genus_lower $x
# or in a pipeline:
@list |> map heegaard_genus_lower |> p
```

heisenberg_min

heisenberg_min — batch 32 builtin.

```
my $result = heisenberg_min $x
# or in a pipeline:
@list |> map heisenberg_min |> p
```

hellinger_distance_step

hellinger_distance_step — batch 40 builtin.

```
my $result = hellinger_distance_step $x
# or in a pipeline:
@list |> map hellinger_distance_step |> p
```

hellinger_kernel

hellinger_kernel — batch 40 builtin.

```
my $result = hellinger_kernel $x
# or in a pipeline:
@list |> map hellinger_kernel |> p
```

helmholtz_capacitance

helmholtz_capacitance — batch 38 builtin.

```
my $result = helmholtz_capacitance $x
# or in a pipeline:
@list |> map helmholtz_capacitance |> p
```

helmholtz_decomp_step

helmholtz_decomp_step — batch 42 builtin.

```
my $result = helmholtz_decomp_step $x
# or in a pipeline:
@list |> map helmholtz_decomp_step |> p
```

henderson_base

`henderson_base` — batch 21 builtin.

```
my $result = henderson_base $x
# or in a pipeline:
@list |> map henderson_base |> p
```

henderson_buffer

`henderson_buffer` — batch 63 builtin.

```
my $result = henderson_buffer $x
# or in a pipeline:
@list |> map henderson_buffer |> p
```

henderson_hasselbalch_solve

`henderson_hasselbalch_solve` — batch 38 builtin.

```
my $result = henderson_hasselbalch_solve $x
# or in a pipeline:
@list |> map henderson_hasselbalch_solve |> p
```

henikoff_weight

`henikoff_weight` — batch 33 builtin.

```
my $result = henikoff_weight $x
# or in a pipeline:
@list |> map henikoff_weight |> p
```

herd_immunity_threshold

`herd_immunity_threshold` — batch 22 builtin.

```
my $result = herd_immunity_threshold $x
# or in a pipeline:
@list |> map herd_immunity_threshold |> p
```

heron

`heron` — geometry / physics builtin. Alias for `heron_area`.

```
my $result = heron $x
# or in a pipeline:
@list |> map heron |> p
```

heron_area

`heron_area` — geometry / physics builtin.

```
my $result = heron_area $x
# or in a pipeline:
@list |> map heron_area |> p
```

heston_price_simple

`heston_price_simple` — batch 20 builtin.

```
my $result = heston_price_simple $x
# or in a pipeline:
@list |> map heston_price_simple |> p
```

heteroskedasticity_test

`heteroskedasticity_test` — batch 36 builtin.

```
my $result = heteroskedasticity_test $x
# or in a pipeline:
@list |> map heteroskedasticity_test |> p
```

heterozygosity

`heterozygosity` — batch 33 builtin.

```
my $result = heterozygosity $x
# or in a pipeline:
@list |> map heterozygosity |> p
```

heun_euler_step

`heun_euler_step` — batch 34 builtin.

```
my $result = heun_euler_step $x
# or in a pipeline:
@list |> map heun_euler_step |> p
```

heun_method

`heun_method` — batch 19 builtin.

```
my $result = heun_method $x
# or in a pipeline:
@list |> map heun_method |> p
```

heun_sde_step

`heun_sde_step` — batch 34 builtin.

```
my $result = heun_sde_step $x
# or in a pipeline:
@list |> map heun_sde_step |> p
```

hex_lower

`hex_lower` — file stat / path builtin.

```
my $result = hex_lower $x
# or in a pipeline:
@list |> map hex_lower |> p
```

hex_of

`hex_of` — base conversion builtin. Alias for `to_hex`.

```
my $result = hex_of $x
# or in a pipeline:
@list |> map hex_of |> p
```

hex_to_bytes

`hex_to_bytes` — string helpers (batch 4) builtin.

```
my $result = hex_to_bytes $input
```

hex_to_rgb

`hex_to_rgb` — color / ansi builtin.

```
my $result = hex_to_rgb $input
```

hex_upper

`hex_upper` — file stat / path builtin.

```
my $result = hex_upper $x
# or in a pipeline:
@list |> map hex_upper |> p
```

hget

`hget` — batch 71 builtin.

```
my $result = hget $x
# or in a pipeline:
@list |> map hget |> p
```

hgetall

`hgetall` — batch 71 builtin.

```
my $result = hgetall $x
# or in a pipeline:
@list |> map hgetall |> p
```

hhi_concentration

`hhi_concentration` — batch 73 builtin.

```
my $result = hhi_concentration $x
# or in a pipeline:
@list |> map hhi_concentration |> p
```

hicksian_demand

`hicksian_demand` — batch 73 builtin.

```
my $result = hicksian_demand $x
# or in a pipeline:
@list |> map hicksian_demand |> p
```

hidden

`hidden` — color / ansi builtin. Alias for `red`.

```
my $result = hidden $x
# or in a pipeline:
@list |> map hidden |> p
```

highpass_filter

`highpass_filter(\@signal, $cutoff)` (alias `hpf`) — applies a simple high-pass filter to remove low-frequency components (DC offset, drift). Signal = original - lowpass.

```
my @with_dc = map { 5 + sin(_/10) } 1:100
my $ac_only = hpf(\@with_dc, 0.1)
```

highway_hash

`highway_hash` — batch 81 builtin.

```
my $result = highway_hash $x
# or in a pipeline:
@list |> map highway_hash |> p
```

hilbert_envelope

`hilbert_envelope` — batch 25 builtin.

```
my $result = hilbert_envelope $x
# or in a pipeline:
@list |> map hilbert_envelope |> p
```

hilbert_signal

`hilbert_signal` — batch 74 builtin.

```
my $result = hilbert_signal $x
# or in a pipeline:
@list |> map hilbert_signal |> p
```

hill_equation

`hill_equation` — batch 12 builtin.

```
my $result = hill_equation $x
# or in a pipeline:
@list |> map hill_equation |> p
```

hill_radius

`hill_radius` — batch 12 builtin.

```
my $result = hill_radius $x
# or in a pipeline:
@list |> map hill_radius |> p
```

hill_sphere_radius

`hill_sphere_radius` — batch 55 builtin.

```
my $result = hill_sphere_radius $x
# or in a pipeline:
@list |> map hill_sphere_radius |> p
```

hill_velocity

`hill_velocity` — batch 31 builtin.

```
my $result = hill_velocity $x
# or in a pipeline:
@list |> map hill_velocity |> p
```

hincrby

`hincrby` — batch 71 builtin.

```
my $result = hincrby $x
# or in a pipeline:
@list |> map hincrby |> p
```

hindley_milner_step

`hindley_milner_step` — batch 67 builtin.

```
my $result = hindley_milner_step $x
# or in a pipeline:
@list |> map hindley_milner_step |> p
```

hindu_lunisolar_month

`hindu_lunisolar_month` — batch 52 builtin.

```
my $result = hindu_lunisolar_month $x
# or in a pipeline:
@list |> map hindu_lunisolar_month |> p
```

hindu_solar_year

`hindu_solar_year` — batch 52 builtin.

```
my $result = hindu_solar_year $x
# or in a pipeline:
@list |> map hindu_solar_year |> p
```

hinge_loss

`hinge_loss` — batch 27 builtin.

```
my $result = hinge_loss $x
# or in a pipeline:
@list |> map hinge_loss |> p
```

hirschberg_lcs_length

`hirschberg_lcs_length` — batch 33 builtin.

```
my $result = hirschberg_lcs_length $x
# or in a pipeline:
@list |> map hirschberg_lcs_length |> p
```

hirzebruch_signature

`hirzebruch_signature` — batch 37 builtin.

```
my $result = hirzebruch_signature $x
# or in a pipeline:
@list |> map hirzebruch_signature |> p
```

hist

`hist` — matrix operations (uncategorized batch) builtin. Alias for `histogram`.

```
my $result = hist $x
# or in a pipeline:
@list |> map hist |> p
```

histogram

`histogram` — matrix operations (uncategorized batch) builtin.

```
my $result = histogram $x
# or in a pipeline:
@list |> map histogram |> p
```

histogram_bin_edges

`histogram_bin_edges` — batch 72 builtin.

```
my $result = histogram_bin_edges $x
# or in a pipeline:
@list |> map histogram_bin_edges |> p
```

histogram_bins

`histogram_bins` — list helpers (batch 4) builtin.

```
my $result = histogram_bins $x
# or in a pipeline:
@list |> map histogram_bins |> p
```

hits_authority

`hits_authority` — batch 75 builtin.

```
my $result = hits_authority $x
# or in a pipeline:
@list |> map hits_authority |> p
```

hits_hub

`hits_hub` — batch 75 builtin.

```
my $result = hits_hub $x
# or in a pipeline:
@list |> map hits_hub |> p
```

hits_simple

`hits_simple` — batch 24 builtin.

```
my $result = hits_simple $x
# or in a pipeline:
@list |> map hits_simple |> p
```

hitting_set_greedy

`hitting_set_greedy` — batch 41 builtin.

```
my $result = hitting_set_greedy $x
# or in a pipeline:
@list |> map hitting_set_greedy |> p
```

hk

`hk` — hash helpers builtin. Alias for `has_key`.

```
my $result = hk $x
# or in a pipeline:
@list |> map hk |> p
```

hkdf_expand_step

`hkdf_expand_step` — batch 26 builtin.

```
my $result = hkdf_expand_step $x
# or in a pipeline:
@list |> map hkdf_expand_step |> p
```

hkeys

`hkeys` — batch 71 builtin.

```
my $result = hkeys $x
# or in a pipeline:
@list |> map hkeys |> p
```

hlen

`hlen` — batch 71 builtin.

```
my $result = hlen $x
# or in a pipeline:
@list |> map hlen |> p
```

hlookup

`hlookup` — batch 57 builtin.

```
my $result = hlookup $x
# or in a pipeline:
@list |> map hlookup |> p
```

hmac_step_xor

`hmac_step_xor` — batches 17-18 builtin.

```
my $result = hmac_step_xor $x
# or in a pipeline:
@list |> map hmac_step_xor |> p
```

hmset

`hmset` — batch 71 builtin.

```
my $result = hmset $x
# or in a pipeline:
@list |> map hmset |> p
```

hochschild_zero

hochschild_zero — batch 37 builtin.

```
my $result = hochschild_zero $x
# or in a pipeline:
@list |> map hochschild_zero |> p
```

hodge_diamond_value

hodge_diamond_value — batch 37 builtin.

```
my $result = hodge_diamond_value $x
# or in a pipeline:
@list |> map hodge_diamond_value |> p
```

hodge_inner_product_one

hodge_inner_product_one — batch 39 builtin.

```
my $result = hodge_inner_product_one $x
# or in a pipeline:
@list |> map hodge_inner_product_one |> p
```

hodge_polynomial_eval

hodge_polynomial_eval — batch 37 builtin.

```
my $result = hodge_polynomial_eval $x
# or in a pipeline:
@list |> map hodge_polynomial_eval |> p
```

hodge_star_one_form

hodge_star_one_form — batch 39 builtin.

```
my $result = hodge_star_one_form $x
# or in a pipeline:
@list |> map hodge_star_one_form |> p
```

hodges_lehmann

hodges_lehmann — batches 14-16 builtin.

```
my $result = hodges_lehmann $x
# or in a pipeline:
@list |> map hodges_lehmann |> p
```

hog_features

hog_features — batch 77 builtin.

```
my $result = hog_features $x
# or in a pipeline:
@list |> map hog_features |> p
```

hohmann_dv1

hohmann_dv1 — batches 14-16 builtin.

```
my $result = hohmann_dv1 $x
# or in a pipeline:
@list |> map hohmann_dv1 |> p
```

hohmann_dv2

hohmann_dv2 — batches 14-16 builtin.

```
my $result = hohmann_dv2 $x
# or in a pipeline:
@list |> map hohmann_dv2 |> p
```

hohmann_total

hohmann_total — batches 14-16 builtin.

```
my $result = hohmann_total $x
# or in a pipeline:
@list |> map hohmann_total |> p
```

holling_type1

holling_type1 — batch 22 builtin.

```
my $result = holling_type1 $x
# or in a pipeline:
@list |> map holling_type1 |> p
```

holling_type2

holling_type2 — batch 22 builtin.

```
my $result = holling_type2 $x
# or in a pipeline:
@list |> map holling_type2 |> p
```

holling_type3

holling_type3 — batch 22 builtin.

```
my $result = holling_type3 $x
# or in a pipeline:
@list |> map holling_type3 |> p
```

holographic_entanglement_step

holographic_entanglement_step — batch 39 builtin.

```
my $result = holographic_entanglement_step $x
# or in a pipeline:
@list |> map holographic_entanglement_step |> p
```

holt_winters_additive

holt_winters_additive — batch 78 builtin.

```
my $result = holt_winters_additive $x
# or in a pipeline:
@list |> map holt_winters_additive |> p
```

holt_winters_multiplicative

holt_winters_multiplicative — batch 78 builtin.

```
my $result = holt_winters_multiplicative $x
# or in a pipeline:
@list |> map holt_winters_multiplicative |> p
```

home_dir

`home_dir` — system introspection builtin.

```
my $result = home_dir $x
# or in a pipeline:
@list |> map home_dir |> p
```

homfly_evaluation

`homfly_evaluation` — batch 37 builtin.

```
my $result = homfly_evaluation $x
# or in a pipeline:
@list |> map homfly_evaluation |> p
```

homology_rank

`homology_rank` — batch 37 builtin.

```
my $result = homology_rank $x
# or in a pipeline:
@list |> map homology_rank |> p
```

homotopy_group_sphere_pi

`homotopy_group_sphere_pi` — batch 37 builtin.

```
my $result = homotopy_group_sphere_pi $x
# or in a pipeline:
@list |> map homotopy_group_sphere_pi |> p
```

hooke

`hooke` — physics formulas builtin. Alias for `spring_force`.

```
my $result = hooke $x
# or in a pipeline:
@list |> map hooke |> p
```

hopcroft_karp

`hopcroft_karp` — batch 24 builtin.

```
my $result = hopcroft_karp $x
# or in a pipeline:
@list |> map hopcroft_karp |> p
```

hopcroft_karp_phase

`hopcroft_karp_phase` — batch 41 builtin.

```
my $result = hopcroft_karp_phase $x
# or in a pipeline:
@list |> map hopcroft_karp_phase |> p
```

hopcroft_karp_step

`hopcroft_karp_step` — batch 75 builtin.

```
my $result = hopcroft_karp_step $x
# or in a pipeline:
@list |> map hopcroft_karp_step |> p
```

horizon_distance

horizon_distance — batches 14-16 builtin.

```
my $result = horizon_distance $x
# or in a pipeline:
@list |> map horizon_distance |> p
```

horn_schunck_step

horn_schunck_step — batch 40 builtin.

```
my $result = horn_schunck_step $x
# or in a pipeline:
@list |> map horn_schunck_step |> p
```

horndeski_step

horndeski_step — batch 39 builtin.

```
my $result = horndeski_step $x
# or in a pipeline:
@list |> map horndeski_step |> p
```

horspool_search

horspool_search — batch 10 builtin.

```
my $result = horspool_search $x
# or in a pipeline:
@list |> map horspool_search |> p
```

hostname_str

hostname_str — more process / system builtin.

```
my $result = hostname_str $x
# or in a pipeline:
@list |> map hostname_str |> p
```

hotelling_location_step

hotelling_location_step — batch 43 builtin.

```
my $result = hotelling_location_step $x
# or in a pipeline:
@list |> map hotelling_location_step |> p
```

hotelling_price_step

hotelling_price_step — batch 43 builtin.

```
my $result = hotelling_price_step $x
# or in a pipeline:
@list |> map hotelling_price_step |> p
```

hough_accumulator_step

`hough_accumulator_step` — batch 40 builtin.

```
my $result = hough_accumulator_step $x
# or in a pipeline:
@list |> map hough_accumulator_step |> p
```

hough_circles

`hough_circles` — batch 77 builtin.

```
my $result = hough_circles $x
# or in a pipeline:
@list |> map hough_circles |> p
```

hough_lines

`hough_lines` — batch 77 builtin.

```
my $result = hough_lines $x
# or in a pipeline:
@list |> map hough_lines |> p
```

hours_to_days

`hours_to_days` — unit conversions builtin.

```
my $result = hours_to_days $input
```

hours_to_minutes

`hours_to_minutes` — unit conversions builtin.

```
my $result = hours_to_minutes $input
```

hours_to_seconds

`hours_to_seconds` — unit conversions builtin.

```
my $result = hours_to_seconds $input
```

householder_root

`householder_root` — batch 19 builtin.

```
my $result = householder_root $x
# or in a pipeline:
@list |> map householder_root |> p
```

hp_to_watts

`hp_to_watts` — file stat / path builtin.

```
my $result = hp_to_watts $input
```

hr

`hr` — data / network builtin. Alias for `http_request`.

```
my $result = hr $x
# or in a pipeline:
@list |> map hr |> p
```

hset

`hset` — batch 71 builtin.

```
my $result = hset $x
# or in a pipeline:
@list |> map hset |> p
```

hsl2rgb

`hsl2rgb` — color operations builtin. Alias for `hsl_to_rgb`.

```
my $result = hsl2rgb $x
# or in a pipeline:
@list |> map hsl2rgb |> p
```

hsl_to_rgb

`hsl_to_rgb` — color operations builtin.

```
my $result = hsl_to_rgb $input
```

hstore_to_array

`hstore_to_array` — batch 70 builtin.

```
my $result = hstore_to_array $input
```

hsv2rgb

`hsv2rgb` — color operations builtin. Alias for `hsv_to_rgb`.

```
my $result = hsv2rgb $x
# or in a pipeline:
@list |> map hsv2rgb |> p
```

hsv_to_rgb

`hsv_to_rgb` — color operations builtin.

```
my $result = hsv_to_rgb $input
```

html_decode

`html_decode` — extended stdlib builtin.

```
my $result = html_decode $x
# or in a pipeline:
@list |> map html_decode |> p
```

html_encode

`html_encode` — extended stdlib builtin.

```
my $result = html_encode $x
# or in a pipeline:
@list |> map html_encode |> p
```

htmld

`htmld` — extended stdlib builtin. Alias for `html_decode`.

```
my $result = htmld $x
# or in a pipeline:
@list |> map htmld |> p
```

htmle

`htmle` — extended stdlib builtin. Alias for `html_encode`.

```
my $result = htmle $x
# or in a pipeline:
@list |> map htmle |> p
```

hubble_constant

`hubble_constant` — physics constants builtin.

```
my $result = hubble_constant $x
# or in a pipeline:
@list |> map hubble_constant |> p
```

hubble_distance_mpc

`hubble_distance_mpc` — batch 30 builtin.

```
my $result = hubble_distance_mpc $x
# or in a pipeline:
@list |> map hubble_distance_mpc |> p
```

hubble_distance_si

`hubble_distance_si` — batch 31 builtin.

```
my $result = hubble_distance_si $x
# or in a pipeline:
@list |> map hubble_distance_si |> p
```

hubble_lcdm

`hubble_lcdm` — batch 31 builtin.

```
my $result = hubble_lcdm $x
# or in a pipeline:
@list |> map hubble_lcdm |> p
```

hubble_time

`hubble_time` — batch 31 builtin.

```
my $result = hubble_time $x
# or in a pipeline:
@list |> map hubble_time |> p
```

huber_m_estimator

`huber_m_estimator` — batches 14-16 builtin.

```
my $result = huber_m_estimator $x
# or in a pipeline:
@list |> map huber_m_estimator |> p
```

huffman_balanced_step

`huffman_balanced_step` — batch 40 builtin.

```
my $result = huffman_balanced_step $x
# or in a pipeline:
@list |> map huffman_balanced_step |> p
```

hull_white_drift

`hull_white_drift` — batch 20 builtin.

```
my $result = hull_white_drift $x
# or in a pipeline:
@list |> map hull_white_drift |> p
```

human_bytes

`human_bytes` — file stat / path builtin. Alias for `format_bytes`.

```
my $result = human_bytes $x
# or in a pipeline:
@list |> map human_bytes |> p
```

human_duration

`human_duration` — file stat / path builtin. Alias for `format_duration`.

```
my $result = human_duration $x
# or in a pipeline:
@list |> map human_duration |> p
```

humidex

`humidex` — batch 12 builtin.

```
my $result = humidex $x
# or in a pipeline:
@list |> map humidex |> p
```

hungarian_method_step

`hungarian_method_step` — batch 41 builtin.

```
my $result = hungarian_method_step $x
# or in a pipeline:
@list |> map hungarian_method_step |> p
```

hungarian_step

`hungarian_step` — batch 75 builtin.

```
my $result = hungarian_step $x
# or in a pipeline:
@list |> map hungarian_step |> p
```

hurst_exponent_estimate

`hurst_exponent_estimate` — batch 42 builtin.

```
my $result = hurst_exponent_estimate $x
# or in a pipeline:
@list |> map hurst_exponent_estimate |> p
```

hvals

`hvals` — batch 71 builtin.

```
my $result = hvals $x
# or in a pipeline:
@list |> map hvals |> p
```

hw_expected_counts

`hw_expected_counts` — batch 33 builtin.

```
my $result = hw_expected_counts $x
# or in a pipeline:
@list |> map hw_expected_counts |> p
```

hxd

`hxd` — crypto / encoding builtin. Alias for `hex_decode`.

```
my $result = hxd $x
# or in a pipeline:
@list |> map hxd |> p
```

hxe

`hxe` — crypto / encoding builtin. Alias for `hex_encode`.

```
my $result = hxe $x
# or in a pipeline:
@list |> map hxe |> p
```

hydrogen_energy_n

`hydrogen_energy_n` — batch 32 builtin.

```
my $result = hydrogen_energy_n $x
# or in a pipeline:
@list |> map hydrogen_energy_n |> p
```

hyp

`hyp` — geometry / physics builtin. Alias for `triangle_hypotenuse`.

```
my $result = hyp $x
# or in a pipeline:
@list |> map hyp |> p
```

hyp1f1

`hyp1f1` — batch 72 builtin.

```
my $result = hyp1f1 $x
# or in a pipeline:
@list |> map hyp1f1 |> p
```

hyp2f1

`hyp2f1` — batch 72 builtin.

```
my $result = hyp2f1 $x
# or in a pipeline:
@list |> map hyp2f1 |> p
```

hyper0f1

`hyper0f1` — batch 29 builtin.

```
my $result = hyper0f1 $x
# or in a pipeline:
@list |> map hyper0f1 |> p
```

hyper1f1

`hyper1f1` — batch 29 builtin.

```
my $result = hyper1f1 $x
# or in a pipeline:
@list |> map hyper1f1 |> p
```

hyper2f1

`hyper2f1` — batch 29 builtin.

```
my $result = hyper2f1 $x
# or in a pipeline:
@list |> map hyper2f1 |> p
```

hypot

`hypot` — trivial numeric / predicate builtins builtin.

```
my $result = hypot $x
# or in a pipeline:
@list |> map hypot |> p
```

hz_to_midi

`hz_to_midi` — batch 30 builtin.

```
my $result = hz_to_midi $input
```

i64_max

`i64_max` — math / numeric (uncategorized batch) builtin.

```
my $result = i64_max $x
# or in a pipeline:
@list |> map i64_max |> p
```

i64_min

`i64_min` — math / numeric (uncategorized batch) builtin.

```
my $result = i64_min $x
# or in a pipeline:
@list |> map i64_min |> p
```

iban_check

`iban_check` — batch 82 builtin.

```
my $result = iban_check $x
# or in a pipeline:
@list |> map iban_check |> p
```

ibnr_estimate

`ibnr_estimate` — batch 60 builtin.

```
my $result = ibnr_estimate $x
# or in a pipeline:
@list |> map ibnr_estimate |> p
```

ical

`ical` — haskell list functions builtin. Alias for `intercalate`.

```
my $result = ical $x
# or in a pipeline:
@list |> map ical |> p
```

ice_age_milankovitch

`ice_age_milankovitch` — batch 42 builtin.

```
my $result = ice_age_milankovitch $x
# or in a pipeline:
@list |> map ice_age_milankovitch |> p
```

icosahedral_number

`icosahedral_number` — batch 11 builtin.

```
my $result = icosahedral_number $x
# or in a pipeline:
@list |> map icosahedral_number |> p
```

id

`id` — trivial numeric / predicate builtins builtin. Alias for `identity`.

```
my $result = id $x
# or in a pipeline:
@list |> map id |> p
```

ida_star

`ida_star` — batch 75 builtin.

```
my $result = ida_star $x
# or in a pipeline:
@list |> map ida_star |> p
```

idamax

`idamax` — batch 66 builtin.

```
my $result = idamax $x
# or in a pipeline:
@list |> map idamax |> p
```

idct

`idct` computes the inverse DCT (Type-III). Reconstructs a signal from DCT coefficients.

```
my @signal = @{idct(dct([1,2,3,4]))}
```

ideal_gas

`ideal_gas` (alias `pv_nrt`) solves PV=nRT for the unknown (pass 0 for the value to solve). Args: P, V, n, T.

```
p pv_nrt(0, 0.0224, 1, 273.15)  # pressure at STP
p pv_nrt(101325, 0, 1, 273.15)   # volume at STP
```

ideal_gas_n

`ideal_gas_n` — batch 21 builtin.

```
my $result = ideal_gas_n $x
# or in a pipeline:
@list |> map ideal_gas_n |> p
```

ideal_gas_volume

`ideal_gas_volume` — physics formulas builtin.

```
my $result = ideal_gas_volume $x
# or in a pipeline:
@list |> map ideal_gas_volume |> p
```

identity

`identity` — trivial numeric / predicate builtins builtin.

```
my $result = identity $x
# or in a pipeline:
@list |> map identity |> p
```

identity_matrix

`identity_matrix` — matrix / linear algebra builtin.

```
my $result = identity_matrix $x
# or in a pipeline:
@list |> map identity_matrix |> p
```

identity_permutation

`identity_permutation` — batch 9 builtin.

```
my $result = identity_permutation $x
# or in a pipeline:
@list |> map identity_permutation |> p
```

idft

`idft(\@spectrum)` — computes the Inverse Discrete Fourier Transform. Takes arrayref of [re, im] pairs and returns the time-domain signal.

```
my $spectrum = dft(\@signal)
my $reconstructed = idft($spectrum)
```

idn_normalize

`idn_normalize` — batch 81 builtin.

```
my $result = idn_normalize $x
# or in a pipeline:
@list |> map idn_normalize |> p
```

idn_to_ascii

`idn_to_ascii` — batch 62 builtin.

```
my $result = idn_to_ascii $input
```

idn_to_unicode

`idn_to_unicode` — batch 62 builtin.

```
my $result = idn_to_unicode $input
```

idt

`idt` — extended stdlib builtin. Alias for `indent`.

```
my $result = idt $x
# or in a pipeline:
@list |> map idt |> p
```

idxb

`idxb` — go/general functional utilities builtin. Alias for `index_by`.

```
my $result = idxb $x
# or in a pipeline:
@list |> map idxb |> p
```

if_else

`if_else` — functional combinators builtin.

```
my $result = if_else $x
# or in a pipeline:
@list |> map if_else |> p
```

if_match_check

`if_match_check` — batch 81 builtin.

```
my $result = if_match_check $x
# or in a pipeline:
@list |> map if_match_check |> p
```

if_none_match_check

`if_none_match_check` — batch 81 builtin.

```
my $result = if_none_match_check $x
# or in a pipeline:
@list |> map if_none_match_check |> p
```

ifr_infection_fatality

`ifr_infection_fatality` — batch 61 builtin.

```
my $result = ifr_infection_fatality $x
# or in a pipeline:
@list |> map ifr_infection_fatality |> p
```

igneous_qapf

`igneous_qapf` — batch 65 builtin.

```
my $result = igneous_qapf $x
# or in a pipeline:
@list |> map igneous_qapf |> p
```

ilerp

`ilerp` — extended stdlib builtin. Alias for `inv_lerp`.

```
my $result = ilerp $x
# or in a pipeline:
@list |> map ilerp |> p
```

image_blur_box

`image_blur_box` — batch 77 builtin.

```
my $result = image_blur_box $x
# or in a pipeline:
@list |> map image_blur_box |> p
```

image_blur_gaussian

`image_blur_gaussian` — batch 77 builtin.

```
my $result = image_blur_gaussian $x
# or in a pipeline:
@list |> map image_blur_gaussian |> p
```

image_brightness

`image_brightness` — batch 77 builtin.

```
my $result = image_brightness $x
# or in a pipeline:
@list |> map image_brightness |> p
```

image_clahe

`image_clahe` — batch 77 builtin.

```
my $result = image_clahe $x
# or in a pipeline:
@list |> map image_clahe |> p
```

image_contrast

`image_contrast` — batch 77 builtin.

```
my $result = image_contrast $x
# or in a pipeline:
@list |> map image_contrast |> p
```

image_dilate

`image_dilate` — batch 77 builtin.

```
my $result = image_dilate $x
# or in a pipeline:
@list |> map image_dilate |> p
```

image_edge_canny

`image_edge_canny` — batch 77 builtin.

```
my $result = image_edge_canny $x
# or in a pipeline:
@list |> map image_edge_canny |> p
```

image_edge_laplacian

`image_edge_laplacian` — batch 77 builtin.

```
my $result = image_edge_laplacian $x
# or in a pipeline:
@list |> map image_edge_laplacian |> p
```

image_edge_sobel

`image_edge_sobel` — batch 77 builtin.

```
my $result = image_edge_sobel $x
# or in a pipeline:
@list |> map image_edge_sobel |> p
```

image_emboss

`image_emboss` — batch 77 builtin.

```
my $result = image_emboss $x
# or in a pipeline:
@list |> map image_emboss |> p
```

image_equalize

`image_equalize` — batch 77 builtin.

```
my $result = image_equalize $x
# or in a pipeline:
@list |> map image_equalize |> p
```

image_erode

`image_erode` — batch 77 builtin.

```
my $result = image_erode $x
# or in a pipeline:
@list |> map image_erode |> p
```

image_flip_h

image_flip_h — batch 77 builtin.

```
my $result = image_flip_h $x
# or in a pipeline:
@list |> map image_flip_h |> p
```

image_flip_v

image_flip_v — batch 77 builtin.

```
my $result = image_flip_v $x
# or in a pipeline:
@list |> map image_flip_v |> p
```

image_gamma

image_gamma — batch 77 builtin.

```
my $result = image_gamma $x
# or in a pipeline:
@list |> map image_gamma |> p
```

image_grayscale

image_grayscale — batch 77 builtin.

```
my $result = image_grayscale $x
# or in a pipeline:
@list |> map image_grayscale |> p
```

image_histogram

image_histogram — batch 77 builtin.

```
my $result = image_histogram $x
# or in a pipeline:
@list |> map image_histogram |> p
```

image_invert

image_invert — batch 77 builtin.

```
my $result = image_invert $x
# or in a pipeline:
@list |> map image_invert |> p
```

image_morphology_close

image_morphology_close — batch 77 builtin.

```
my $result = image_morphology_close $x
# or in a pipeline:
@list |> map image_morphology_close |> p
```

image_morphology_open

image_morphology_open — batch 77 builtin.

```
my $result = image_morphology_open $x
# or in a pipeline:
@list |> map image_morphology_open |> p
```

image_motion_blur

`image_motion_blur` — batch 77 builtin.

```
my $result = image_motion_blur $x
# or in a pipeline:
@list |> map image_motion_blur |> p
```

image_posterize

`image_posterize` — batch 77 builtin.

```
my $result = image_posterize $x
# or in a pipeline:
@list |> map image_posterize |> p
```

image_resize

`image_resize` — batch 77 builtin.

```
my $result = image_resize $x
# or in a pipeline:
@list |> map image_resize |> p
```

image_rotate

`image_rotate` — batch 77 builtin.

```
my $result = image_rotate $x
# or in a pipeline:
@list |> map image_rotate |> p
```

image_sepia

`image_sepia` — batch 77 builtin.

```
my $result = image_sepia $x
# or in a pipeline:
@list |> map image_sepia |> p
```

image_sharpen

`image_sharpen` — batch 77 builtin.

```
my $result = image_sharpen $x
# or in a pipeline:
@list |> map image_sharpen |> p
```

image_solarize

`image_solarize` — batch 77 builtin.

```
my $result = image_solarize $x
# or in a pipeline:
@list |> map image_solarize |> p
```

image_threshold

`image_threshold` — batch 77 builtin.

```
my $result = image_threshold $x
# or in a pipeline:
@list |> map image_threshold |> p
```

imei_check

`imei_check` — batch 82 builtin.

```
my $result = imei_check $x
# or in a pipeline:
@list |> map imei_check |> p
```

imex_euler_step

`imex_euler_step` — batch 34 builtin.

```
my $result = imex_euler_step $x
# or in a pipeline:
@list |> map imex_euler_step |> p
```

imex_theta_split

`imex_theta_split` — batch 34 builtin.

```
my $result = imex_theta_split $x
# or in a pipeline:
@list |> map imex_theta_split |> p
```

impedance_rlc

`impedance_rlc` — physics formulas builtin.

```
my $result = impedance_rlc $x
# or in a pipeline:
@list |> map impedance_rlc |> p
```

impl

`impl` declares which traits a class implements.

```
trait Printable { fn to_str }
class Item impl Printable {
    name: Str
    fn to_str { $self->name }
}
```

Multiple traits: `class X impl A, B, C { }`

implicit_euler

`implicit_euler` — batch 19 builtin. Alias for `backward_euler`.

```
my $result = implicit_euler $x
# or in a pipeline:
@list |> map implicit_euler |> p
```

impulse

`impulse($force, $time)` — compute impulse J = F×t (N□s). Change in momentum.

```
p impulse(100, 0.5)   # 50 N·s
```

impulse_response_n

`impulse_response_n` — batch 74 builtin.

```
my $result = impulse_response_n $x
# or in a pipeline:
@list |> map impulse_response_n |> p
```

impulse_response_step

`impulse_response_step` — batch 36 builtin.

```
my $result = impulse_response_step $x
# or in a pipeline:
@list |> map impulse_response_step |> p
```

imputation_efficient_check

`imputation_efficient_check` — batch 43 builtin.

```
my $result = imputation_efficient_check $x
# or in a pipeline:
@list |> map imputation_efficient_check |> p
```

imputation_individual_rational

`imputation_individual_rational` — batch 43 builtin.

```
my $result = imputation_individual_rational $x
# or in a pipeline:
@list |> map imputation_individual_rational |> p
```

imu_madgwick_step

`imu_madgwick_step` — batch 59 builtin.

```
my $result = imu_madgwick_step $x
# or in a pipeline:
@list |> map imu_madgwick_step |> p
```

imu_mahony_step

`imu_mahony_step` — batch 59 builtin.

```
my $result = imu_mahony_step $x
# or in a pipeline:
@list |> map imu_mahony_step |> p
```

in_degree_directed

`in_degree_directed` — batch 13 builtin.

```
my $result = in_degree_directed $x
# or in a pipeline:
@list |> map in_degree_directed |> p
```

in_range

`in_range` — extended stdlib builtin.

```
my $result = in_range $x
# or in a pipeline:
@list |> map in_range |> p
```

inc

inc — pipeline / string helpers builtin.

```
my $result = inc $x
# or in a pipeline:
@list |> map inc |> p
```

inc_each

inc_each — trivial numeric helpers (batch 4) builtin.

```
my $result = inc_each $x
# or in a pipeline:
@list |> map inc_each |> p
```

incentive_compatibility_check

incentive_compatibility_check — batch 43 builtin.

```
my $result = incentive_compatibility_check $x
# or in a pipeline:
@list |> map incentive_compatibility_check |> p
```

inches_to_cm

inches_to_cm — unit conversions builtin.

```
my $result = inches_to_cm $input
```

incidence_algebra_dim

incidence_algebra_dim — batch 37 builtin.

```
my $result = incidence_algebra_dim $x
# or in a pipeline:
@list |> map incidence_algebra_dim |> p
```

includes_val

includes_val — javascript array/object methods builtin.

```
my $result = includes_val $x
# or in a pipeline:
@list |> map includes_val |> p
```

income_elasticity

income_elasticity — batch 73 builtin.

```
my $result = income_elasticity $x
# or in a pipeline:
@list |> map income_elasticity |> p
```

incr

incr — batch 71 builtin.

```
my $result = incr $x
# or in a pipeline:
@list |> map incr |> p
```

incrby

`incrby` — batch 71 builtin.

```
my $result = incrby $x
# or in a pipeline:
@list |> map incrby |> p
```

incv

`incv` — javascript array/object methods builtin. Alias for `includes_val`.

```
my $result = incv $x
# or in a pipeline:
@list |> map incv |> p
```

indenergy

`indenergy` — physics formulas builtin. Alias for `inductor_energy`.

```
my $result = indenergy $x
# or in a pipeline:
@list |> map indenergy |> p
```

indent

`indent` — extended stdlib builtin.

```
my $result = indent $x
# or in a pipeline:
@list |> map indent |> p
```

indent_text

`indent_text` — string processing (uncategorized batch) builtin.

```
my $result = indent_text $x
# or in a pipeline:
@list |> map indent_text |> p
```

independence_number_upper

`independence_number_upper` — batch 41 builtin.

```
my $result = independence_number_upper $x
# or in a pipeline:
@list |> map independence_number_upper |> p
```

index_by

`index_by` — go/general functional utilities builtin.

```
my $result = index_by $x
# or in a pipeline:
@list |> map index_by |> p
```

index_calculus_naive

`index_calculus_naive` — batch 35 builtin.

```
my $result = index_calculus_naive $x
# or in a pipeline:
@list |> map index_calculus_naive |> p
```

index_from_speed

index_from_speed — batch 23 builtin.

```
my $result = index_from_speed $x
# or in a pipeline:
@list |> map index_from_speed |> p
```

index_match

index_match — batch 57 builtin.

```
my $result = index_match $x
# or in a pipeline:
@list |> map index_match |> p
```

index_of

index_of — collection (batch 2) builtin.

```
my $result = index_of $x
# or in a pipeline:
@list |> map index_of |> p
```

index_of_coincidence

index_of_coincidence — batch 26 builtin.

```
my $result = index_of_coincidence $x
# or in a pipeline:
@list |> map index_of_coincidence |> p
```

index_of_elem

index_of_elem — list helpers (batch 4) builtin.

```
my $result = index_of_elem $x
# or in a pipeline:
@list |> map index_of_elem |> p
```

indexes_of

indexes_of — string (batch 2) builtin.

```
my $result = indexes_of $x
# or in a pipeline:
@list |> map indexes_of |> p
```

indian_monsoon_index

indian_monsoon_index — batch 42 builtin.

```
my $result = indian_monsoon_index $x
# or in a pipeline:
@list |> map indian_monsoon_index |> p
```

indirect

`indirect` — batch 57 builtin.

```
my $result = indirect $x
# or in a pipeline:
@list |> map indirect |> p
```

indirect_age_adjusted

`indirect_age_adjusted` — batch 61 builtin.

```
my $result = indirect_age_adjusted $x
# or in a pipeline:
@list |> map indirect_age_adjusted |> p
```

indirect_utility

`indirect_utility` — batch 73 builtin.

```
my $result = indirect_utility $x
# or in a pipeline:
@list |> map indirect_utility |> p
```

indivisible_envy_free_check

`indivisible_envy_free_check` — batch 43 builtin.

```
my $result = indivisible_envy_free_check $x
# or in a pipeline:
@list |> map indivisible_envy_free_check |> p
```

inductance_parallel

`inductance_parallel` — batch 12 builtin.

```
my $result = inductance_parallel $x
# or in a pipeline:
@list |> map inductance_parallel |> p
```

inductance_series

`inductance_series` — batch 12 builtin.

```
my $result = inductance_series $x
# or in a pipeline:
@list |> map inductance_series |> p
```

inductor_energy

`inductor_energy` — physics formulas builtin.

```
my $result = inductor_energy $x
# or in a pipeline:
@list |> map inductor_energy |> p
```

infinite_well_energy

`infinite_well_energy` — batch 32 builtin.

```
my $result = infinite_well_energy $x
# or in a pipeline:
@list |> map infinite_well_energy |> p
```

information_bottleneck_step

`information_bottleneck_step` — batch 40 builtin.

```
my $result = information_bottleneck_step $x
# or in a pipeline:
@list |> map information_bottleneck_step |> p
```

information_gain

`information_gain` — batch 27 builtin.

```
my $result = information_gain $x
# or in a pipeline:
@list |> map information_gain |> p
```

information_ratio

`information_ratio` — batch 13 builtin.

```
my $result = information_ratio $x
# or in a pipeline:
@list |> map information_ratio |> p
```

init_list

`init_list` — list helpers (batch 4) builtin.

```
my $result = init_list $x
# or in a pipeline:
@list |> map init_list |> p
```

initial_bearing

`initial_bearing` — batch 28 builtin.

```
my $result = initial_bearing $x
# or in a pipeline:
@list |> map initial_bearing |> p
```

initials

`initials` — string processing (uncategorized batch) builtin.

```
my $result = initials $x
# or in a pipeline:
@list |> map initials |> p
```

inits

`inits` — additional missing stdlib functions builtin.

```
my $result = inits $x
# or in a pipeline:
@list |> map inits |> p
```

inits_str

`inits_str` — string processing (uncategorized batch) builtin. Alias for `initials`.

```
my $result = inits_str $x
# or in a pipeline:
@list |> map inits_str |> p
```

inject

`inject` — functional / iterator builtin.

```
my $result = inject $x
# or in a pipeline:
@list |> map inject |> p
```

inrng

`inrng` — extended stdlib builtin. Alias for `in_range`.

```
my $result = inrng $x
# or in a pipeline:
@list |> map inrng |> p
```

insa

`insa` — go/general functional utilities builtin. Alias for `insert_at`.

```
my $result = insa $x
# or in a pipeline:
@list |> map insa |> p
```

insert_at

`insert_at` — go/general functional utilities builtin.

```
my $result = insert_at $x
# or in a pipeline:
@list |> map insert_at |> p
```

insert_at_idx

`insert_at_idx` — batch 9 builtin.

```
my $result = insert_at_idx $x
# or in a pipeline:
@list |> map insert_at_idx |> p
```

insert_sorted

`insert_sorted` — additional missing stdlib functions builtin.

```
my $result = insert_sorted $x
# or in a pipeline:
@list |> map insert_sorted |> p
```

insert_str

`insert_str` — string helpers (batch 4) builtin.

```
my $result = insert_str $x
# or in a pipeline:
@list |> map insert_str |> p
```

insp

`insp` — rust iterator methods builtin. Alias for `inspect`.

```
my $result = insp $x
# or in a pipeline:
@list |> map insp |> p
```

inspect

`inspect` — rust iterator methods builtin.

```
my $result = inspect $x
# or in a pipeline:
@list |> map inspect |> p
```

inssrt

`inssrt` — additional missing stdlib functions builtin. Alias for `insert_sorted`.

```
my $result = inssrt $x
# or in a pipeline:
@list |> map inssrt |> p
```

int_bits

`int_bits` — math functions builtin.

```
my $result = int_bits $x
# or in a pipeline:
@list |> map int_bits |> p
```

int_to_ipv4

`int_to_ipv4` — network / validation builtin.

```
my $result = int_to_ipv4 $input
```

int_to_roman

`int_to_roman` — roman numerals builtin.

```
my $result = int_to_roman $input
```

integral_length_scale

`integral_length_scale` — batch 42 builtin.

```
my $result = integral_length_scale $x
# or in a pipeline:
@list |> map integral_length_scale |> p
```

intercalate

`intercalate` — haskell list functions builtin.

```
my $result = intercalate $x
# or in a pipeline:
@list |> map intercalate |> p
```

interleave_lists

`interleave_lists` — list helpers (batch 4) builtin.

```
my $result = interleave_lists $x
# or in a pipeline:
@list |> map interleave_lists |> p
```

internal_rate_of_return

`internal_rate_of_return` — batch 10 builtin.

```
my $result = internal_rate_of_return $x
# or in a pipeline:
@list |> map internal_rate_of_return |> p
```

interp1d

`interp1d` — batch 72 builtin.

```
my $result = interp1d $x
# or in a pipeline:
@list |> map interp1d |> p
```

interpose

`interpose` — go/general functional utilities builtin.

```
my $result = interpose $x
# or in a pipeline:
@list |> map interpose |> p
```

intersect_list

`intersect_list` — additional missing stdlib functions builtin.

```
my $result = intersect_list $x
# or in a pipeline:
@list |> map intersect_list |> p
```

intersperse

`intersperse` — collection helpers (trivial) builtin. Alias for `riffle`.

```
my $result = intersperse $x
# or in a pipeline:
@list |> map intersperse |> p
```

intersperse_char

`intersperse_char` — string helpers (batch 4) builtin.

```
my $result = intersperse_char $x
# or in a pipeline:
@list |> map intersperse_char |> p
```

intersperse_val

`intersperse_val` — haskell list functions builtin.

```
my $result = intersperse_val $x
# or in a pipeline:
@list |> map intersperse_val |> p
```

intersperse_with

`intersperse_with` — additional missing stdlib functions builtin.

```
my $result = intersperse_with $x
# or in a pipeline:
@list |> map intersperse_with |> p
```

interval_quality_size

`interval_quality_size` — batch 64 builtin.

```
my $result = interval_quality_size $x
# or in a pipeline:
@list |> map interval_quality_size |> p
```

interval_semitones

`interval_semitones` — batch 9 builtin.

```
my $result = interval_semitones $x
# or in a pipeline:
@list |> map interval_semitones |> p
```

intl

`intl` — additional missing stdlib functions builtin. Alias for `intersect_list`.

```
my $result = intl $x
# or in a pipeline:
@list |> map intl |> p
```

into

`into` — algebraic match builtin.

```
my $result = into $x
# or in a pipeline:
@list |> map into |> p
```

intrate

`intrate` — batch 81 builtin.

```
my $result = intrate $x
# or in a pipeline:
@list |> map intrate |> p
```

intrinsic_growth_rate

`intrinsic_growth_rate` — batch 22 builtin.

```
my $result = intrinsic_growth_rate $x
# or in a pipeline:
@list |> map intrinsic_growth_rate |> p
```

intrinsic_principal_curv

intrinsic_principal_curv — batch 39 builtin.

```
my $result = intrinsic_principal_curv $x
# or in a pipeline:
@list |> map intrinsic_principal_curv |> p
```

inv_lerp

inv_lerp — extended stdlib builtin.

```
my $result = inv_lerp $x
# or in a pipeline:
@list |> map inv_lerp |> p
```

inverse

inverse — trivial numeric / predicate builtins builtin.

```
my $result = inverse $x
# or in a pipeline:
@list |> map inverse |> p
```

inverse_chi2_pdf

inverse_chi2_pdf — batches 17-18 builtin.

```
my $result = inverse_chi2_pdf $x
# or in a pipeline:
@list |> map inverse_chi2_pdf |> p
```

inverse_kinematics_2link

inverse_kinematics_2link — batch 59 builtin.

```
my $result = inverse_kinematics_2link $x
# or in a pipeline:
@list |> map inverse_kinematics_2link |> p
```

inverse_lerp

inverse_lerp — math formulas builtin.

```
my $result = inverse_lerp $x
# or in a pipeline:
@list |> map inverse_lerp |> p
```

inverse_qft

inverse_qft — batch 80 builtin.

```
my $result = inverse_qft $x
# or in a pipeline:
@list |> map inverse_qft |> p
```

inverse_quad_interp

inverse_quad_interp — batch 19 builtin.

```
my $result = inverse_quad_interp $x
# or in a pipeline:
@list |> map inverse_quad_interp |> p
```

inverse_simpson

`inverse_simpson` — batch 22 builtin.

```
my $result = inverse_simpson $x
# or in a pipeline:
@list |> map inverse_simpson |> p
```

inverse_weibull_pdf

`inverse_weibull_pdf` — batches 17-18 builtin.

```
my $result = inverse_weibull_pdf $x
# or in a pipeline:
@list |> map inverse_weibull_pdf |> p
```

invert

`invert` — hash helpers builtin.

```
my $result = invert $x
# or in a pipeline:
@list |> map invert |> p
```

invert_perm

`invert_perm` — batch 54 builtin.

```
my $result = invert_perm $x
# or in a pipeline:
@list |> map invert_perm |> p
```

ioctl

`ioctl` — ipc builtin.

```
my $result = ioctl $x
# or in a pipeline:
@list |> map ioctl |> p
```

ionic_conductivity_arrhenius

`ionic_conductivity_arrhenius` — batch 38 builtin.

```
my $result = ionic_conductivity_arrhenius $x
# or in a pipeline:
@list |> map ionic_conductivity_arrhenius |> p
```

ionic_liquid_viscosity_step

`ionic_liquid_viscosity_step` — batch 38 builtin.

```
my $result = ionic_liquid_viscosity_step $x
# or in a pipeline:
@list |> map ionic_liquid_viscosity_step |> p
```

ionic_strength

`ionic_strength` — batch 12 builtin.

```
my $result = ionic_strength $x
# or in a pipeline:
@list |> map ionic_strength |> p
```

iota_n

`iota_n` — batch 54 builtin.

```
my $result = iota_n $x
# or in a pipeline:
@list |> map iota_n |> p
```

iou_2d_axis_aligned

`iou_2d_axis_aligned` — batches 17-18 builtin.

```
my $result = iou_2d_axis_aligned $x
# or in a pipeline:
@list |> map iou_2d_axis_aligned |> p
```

ipmt

`ipmt` — batch 57 builtin.

```
my $result = ipmt $x
# or in a pipeline:
@list |> map ipmt |> p
```

ipos

`ipos` — go/general functional utilities builtin. Alias for `interpose`.

```
my $result = ipos $x
# or in a pipeline:
@list |> map ipos |> p
```

ipv4_to_int

`ipv4_to_int` — network / validation builtin.

```
my $result = ipv4_to_int $input
```

ipv_potential_vorticity

`ipv_potential_vorticity` — batch 42 builtin.

```
my $result = ipv_potential_vorticity $x
# or in a pipeline:
@list |> map ipv_potential_vorticity |> p
```

iqr

`iqr` — extended stdlib builtin.

```
my $result = iqr $x
# or in a pipeline:
@list |> map iqr |> p
```

irr

`irr(@cashflows)` — computes the Internal Rate of Return for a series of cash flows. First value is typically negative (initial investment). Uses Newton-Raphson iteration.

```
my @cf = (-1000, 300, 420, 680)
p irr(@cf)   # ~0.166 (16.6%)
```

is_7_smooth

`is_7_smooth` — batch 10 builtin. Alias for `seven_smooth_q`.

```
my $result = is_7_smooth $x
# or in a pipeline:
@list |> map is_7_smooth |> p
```

is_abundant

`is_abundant` — number theory / primes builtin.

```
my $result = is_abundant $x
# or in a pipeline:
@list |> map is_abundant |> p
```

is_alnum

`is_alnum` — trivial string ops builtin.

```
my $result = is_alnum $x
# or in a pipeline:
@list |> map is_alnum |> p
```

is_alpha

`is_alpha` — trivial string ops builtin.

```
my $result = is_alpha $x
# or in a pipeline:
@list |> map is_alpha |> p
```

is_anagram

`is_anagram` — dsp / signal (extended) builtin.

```
my $result = is_anagram $x
# or in a pipeline:
@list |> map is_anagram |> p
```

is_anagram_q

`is_anagram_q` — batch 11 builtin. Alias for `anagram_q`.

```
my $result = is_anagram_q $x
# or in a pipeline:
@list |> map is_anagram_q |> p
```

is_armstrong

`is_armstrong` — batch 11 builtin. Alias for `armstrong_q`.

```
my $result = is_armstrong $x
# or in a pipeline:
@list |> map is_armstrong |> p
```

is_array

`is_array` — trivial type predicates builtin.

```
my $result = is_array $x
# or in a pipeline:
@list |> map is_array |> p
```

is_arrayref

`is_arrayref` — trivial type predicates builtin. Alias for `is_array`.

```
my $result = is_arrayref $x
# or in a pipeline:
@list |> map is_arrayref |> p
```

is_asc

`is_asc` — predicates (batch 2) builtin. Alias for `is_sorted`.

```
my $result = is_asc $x
# or in a pipeline:
@list |> map is_asc |> p
```

is_ascii

`is_ascii` — extended stdlib builtin.

```
my $result = is_ascii $x
# or in a pipeline:
@list |> map is_ascii |> p
```

is_b_smooth

`is_b_smooth` — batch 35 builtin.

```
my $result = is_b_smooth $x
# or in a pipeline:
@list |> map is_b_smooth |> p
```

is_balanced_parens

`is_balanced_parens` — dsp / signal (extended) builtin.

```
my $result = is_balanced_parens $x
# or in a pipeline:
@list |> map is_balanced_parens |> p
```

is_base64

`is_base64` — extended stdlib builtin.

```
my $result = is_base64 $x
# or in a pipeline:
@list |> map is_base64 |> p
```

is_between

`is_between` — predicates (batch 4) builtin.

```
my $result = is_between $x
# or in a pipeline:
@list |> map is_between |> p
```

is_binary_str

`is_binary_str` — extended stdlib builtin.

```
my $result = is_binary_str $x
# or in a pipeline:
@list |> map is_binary_str |> p
```

is_bipartite_graph

`is_bipartite_graph` — extended stdlib builtin.

```
my $result = is_bipartite_graph $x
# or in a pipeline:
@list |> map is_bipartite_graph |> p
```

is_blank

`is_blank` — trivial string ops builtin.

```
my $result = is_blank $x
# or in a pipeline:
@list |> map is_blank |> p
```

is_blank_or_nil

`is_blank_or_nil` — predicates (batch 4) builtin.

```
my $result = is_blank_or_nil $x
# or in a pipeline:
@list |> map is_blank_or_nil |> p
```

is_callable

`is_callable` — functional combinators builtin.

```
my $result = is_callable $x
# or in a pipeline:
@list |> map is_callable |> p
```

is_carmichael

`is_carmichael` — batch 10 builtin. Alias for `carmichael_q`.

```
my $result = is_carmichael $x
# or in a pipeline:
@list |> map is_carmichael |> p
```

is_code

`is_code` — trivial type predicates builtin.

```
my $result = is_code $x
# or in a pipeline:
@list |> map is_code |> p
```

is_coderef

`is_coderef` — trivial type predicates builtin. Alias for `is_code`.

```
my $result = is_coderef $x
# or in a pipeline:
@list |> map is_coderef |> p
```

is_connected

`is_connected` — batch 13 builtin.

```
my $result = is_connected $x
# or in a pipeline:
@list |> map is_connected |> p
```

is_consonant

`is_consonant` — string (batch 2) builtin.

```
my $result = is_consonant $x
# or in a pipeline:
@list |> map is_consonant |> p
```

is_control

`is_control` — dsp / signal (extended) builtin.

```
my $result = is_control $x
# or in a pipeline:
@list |> map is_control |> p
```

is_coprime

`is_coprime` — extended stdlib batch 3 builtin.

```
my $result = is_coprime $x
# or in a pipeline:
@list |> map is_coprime |> p
```

is_credit_card

`is_credit_card` — extended stdlib builtin.

```
my $result = is_credit_card $x
# or in a pipeline:
@list |> map is_credit_card |> p
```

is_def

`is_def` — trivial type predicates builtin. Alias for `is_defined`.

```
my $result = is_def $x
# or in a pipeline:
@list |> map is_def |> p
```

is_deficient

`is_deficient` — number theory / primes builtin.

```
my $result = is_deficient $x
# or in a pipeline:
@list |> map is_deficient |> p
```

is_defined

`is_defined` — trivial type predicates builtin.

```
my $result = is_defined $x
# or in a pipeline:
@list |> map is_defined |> p
```

is_desc

`is_desc` — predicates (batch 2) builtin. Alias for `is_sorted_desc`.

```
my $result = is_desc $x
# or in a pipeline:
@list |> map is_desc |> p
```

is_digit

`is_digit` — trivial string ops builtin.

```
my $result = is_digit $x
# or in a pipeline:
@list |> map is_digit |> p
```

is_divisible_by

`is_divisible_by` — predicates (batch 4) builtin.

```
my $result = is_divisible_by $x
# or in a pipeline:
@list |> map is_divisible_by |> p
```

is_email

`is_email` — predicates (batch 4) builtin.

```
my $result = is_email $x
# or in a pipeline:
@list |> map is_email |> p
```

is_empty

`is_empty` — trivial string ops builtin.

```
my $result = is_empty $x
# or in a pipeline:
@list |> map is_empty |> p
```

is_empty_arr

`is_empty_arr` — predicates (batch 2) builtin.

```
my $result = is_empty_arr $x
# or in a pipeline:
@list |> map is_empty_arr |> p
```

is_empty_hash

`is_empty_hash` — predicates (batch 2) builtin.

```
my $result = is_empty_hash $x
# or in a pipeline:
@list |> map is_empty_hash |> p
```

is_even

`is_even` — predicates (batch 4) builtin.

```
my $result = is_even $x
# or in a pipeline:
@list |> map is_even |> p
```

is_even_num

`is_even_num` — extended stdlib builtin.

```
my $result = is_even_num $x
# or in a pipeline:
@list |> map is_even_num |> p
```

is_executable

`is_executable` — file stat / path builtin.

```
my $result = is_executable $x
# or in a pipeline:
@list |> map is_executable |> p
```

is_falsy

`is_falsy` — predicates (batch 4) builtin.

```
my $result = is_falsy $x
# or in a pipeline:
@list |> map is_falsy |> p
```

is_fibonacci

`is_fibonacci` — predicates (batch 4) builtin.

```
my $result = is_fibonacci $x
# or in a pipeline:
@list |> map is_fibonacci |> p
```

is_finite

`is_finite` — trig / math (batch 2) builtin.

```
my $result = is_finite $x
# or in a pipeline:
@list |> map is_finite |> p
```

is_float

is_float — trivial type predicates builtin.

```
my $result = is_float $x
# or in a pipeline:
@list |> map is_float |> p
```

is_harshad

is_harshad — extended stdlib batch 3 builtin.

```
my $result = is_harshad $x
# or in a pipeline:
@list |> map is_harshad |> p
```

is_hash

is_hash — trivial type predicates builtin.

```
my $result = is_hash $x
# or in a pipeline:
@list |> map is_hash |> p
```

is_hashref

is_hashref — trivial type predicates builtin. Alias for is_hash.

```
my $result = is_hashref $x
# or in a pipeline:
@list |> map is_hashref |> p
```

is_heterogram

is_heterogram — extended stdlib batch 3 builtin.

```
my $result = is_heterogram $x
# or in a pipeline:
@list |> map is_heterogram |> p
```

is_hex_color

is_hex_color — predicates (batch 4) builtin.

```
my $result = is_hex_color $x
# or in a pipeline:
@list |> map is_hex_color |> p
```

is_hex_str

is_hex_str — extended stdlib builtin.

```
my $result = is_hex_str $x
# or in a pipeline:
@list |> map is_hex_str |> p
```

is_hostname_valid

is_hostname_valid — extended stdlib builtin.

```
my $result = is_hostname_valid $x
# or in a pipeline:
@list |> map is_hostname_valid |> p
```

is_iban

`is_iban` — extended stdlib builtin.

```
my $result = is_iban $x
# or in a pipeline:
@list |> map is_iban |> p
```

is_in_range

`is_in_range` — predicates (batch 4) builtin.

```
my $result = is_in_range $x
# or in a pipeline:
@list |> map is_in_range |> p
```

is_inf

`is_inf` — trig / math (batch 2) builtin. Alias for `is_infinite`.

```
my $result = is_inf $x
# or in a pipeline:
@list |> map is_inf |> p
```

is_infinite

`is_infinite` — trig / math (batch 2) builtin.

```
my $result = is_infinite $x
# or in a pipeline:
@list |> map is_infinite |> p
```

is_int

`is_int` — trivial type predicates builtin.

```
my $result = is_int $x
# or in a pipeline:
@list |> map is_int |> p
```

is_integer

`is_integer` — trivial type predicates builtin. Alias for `is_int`.

```
my $result = is_integer $x
# or in a pipeline:
@list |> map is_integer |> p
```

is_ipv4

`is_ipv4` — predicates (batch 4) builtin.

```
my $result = is_ipv4 $x
# or in a pipeline:
@list |> map is_ipv4 |> p
```

is_ipv4_addr

`is_ipv4_addr` — extended stdlib builtin.

```
my $result = is_ipv4_addr $x
# or in a pipeline:
@list |> map is_ipv4_addr |> p
```

is_ipv6_addr

`is_ipv6_addr` — extended stdlib builtin.

```
my $result = is_ipv6_addr $x
# or in a pipeline:
@list |> map is_ipv6_addr |> p
```

is_isbn10

`is_isbn10` — extended stdlib builtin.

```
my $result = is_isbn10 $x
# or in a pipeline:
@list |> map is_isbn10 |> p
```

is_isbn13

`is_isbn13` — extended stdlib builtin.

```
my $result = is_isbn13 $x
# or in a pipeline:
@list |> map is_isbn13 |> p
```

is_iso_date

`is_iso_date` — extended stdlib builtin.

```
my $result = is_iso_date $x
# or in a pipeline:
@list |> map is_iso_date |> p
```

is_iso_datetime

`is_iso_datetime` — extended stdlib builtin.

```
my $result = is_iso_datetime $x
# or in a pipeline:
@list |> map is_iso_datetime |> p
```

is_iso_time

`is_iso_time` — extended stdlib builtin.

```
my $result = is_iso_time $x
# or in a pipeline:
@list |> map is_iso_time |> p
```

is_isogram

`is_isogram` — extended stdlib batch 3 builtin.

```
my $result = is_isogram $x
# or in a pipeline:
@list |> map is_isogram |> p
```

is_json

`is_json` — extended stdlib builtin.

```
my $result = is_json $x
# or in a pipeline:
@list |> map is_json |> p
```

is_kaprekar

`is_kaprekar` — extended stdlib batch 3 builtin.

```
my $result = is_kaprekar $x
# or in a pipeline:
@list |> map is_kaprekar |> p
```

is_keith

`is_keith` — batch 11 builtin. Alias for `keith_q`.

```
my $result = is_keith $x
# or in a pipeline:
@list |> map is_keith |> p
```

is_leap

`is_leap` — date helpers builtin. Alias for `is_leap_year`.

```
my $result = is_leap $x
# or in a pipeline:
@list |> map is_leap |> p
```

is_leap_year

`is_leap_year` — date helpers builtin.

```
my $result = is_leap_year $x
# or in a pipeline:
@list |> map is_leap_year |> p
```

is_lower

`is_lower` — trivial string ops builtin.

```
my $result = is_lower $x
# or in a pipeline:
@list |> map is_lower |> p
```

is_mac_addr

`is_mac_addr` — extended stdlib builtin.

```
my $result = is_mac_addr $x
# or in a pipeline:
@list |> map is_mac_addr |> p
```

is_match

`is_match` — extended stdlib builtin.

```
my $result = is_match $x
# or in a pipeline:
@list |> map is_match |> p
```

is_monotonic

`is_monotonic` — extended stdlib batch 3 builtin.

```
my $result = is_monotonic $x
# or in a pipeline:
@list |> map is_monotonic |> p
```

is_multiple_of

`is_multiple_of` — predicates (batch 4) builtin.

```
my $result = is_multiple_of $x
# or in a pipeline:
@list |> map is_multiple_of |> p
```

is_nan

`is_nan` — trig / math (batch 2) builtin.

```
my $result = is_nan $x
# or in a pipeline:
@list |> map is_nan |> p
```

is_narcissistic

`is_narcissistic` — extended stdlib batch 3 builtin.

```
my $result = is_narcissistic $x
# or in a pipeline:
@list |> map is_narcissistic |> p
```

is_negative

`is_negative` — predicates (batch 4) builtin.

```
my $result = is_negative $x
# or in a pipeline:
@list |> map is_negative |> p
```

is_negative_num

`is_negative_num` — extended stdlib builtin.

```
my $result = is_negative_num $x
# or in a pipeline:
@list |> map is_negative_num |> p
```

is_nil

`is_nil` — predicates (batch 4) builtin.

```
my $result = is_nil $x
# or in a pipeline:
@list |> map is_nil |> p
```

is_nonzero

is_nonzero — predicates (batch 4) builtin.

```
my $result = is_nonzero $x
# or in a pipeline:
@list |> map is_nonzero |> p
```

is_numeric

is_numeric — trivial string ops builtin.

```
my $result = is_numeric $x
# or in a pipeline:
@list |> map is_numeric |> p
```

is_numeric_string

is_numeric_string — dsp / signal (extended) builtin.

```
my $result = is_numeric_string $x
# or in a pipeline:
@list |> map is_numeric_string |> p
```

is_octal_str

is_octal_str — extended stdlib builtin.

```
my $result = is_octal_str $x
# or in a pipeline:
@list |> map is_octal_str |> p
```

is_odd

is_odd — predicates (batch 4) builtin.

```
my $result = is_odd $x
# or in a pipeline:
@list |> map is_odd |> p
```

is_odd_num

is_odd_num — extended stdlib builtin.

```
my $result = is_odd_num $x
# or in a pipeline:
@list |> map is_odd_num |> p
```

is_pair

is_pair — predicates (batch 2) builtin.

```
my $result = is_pair $x
# or in a pipeline:
@list |> map is_pair |> p
```

is_palindrome

is_palindrome — string (batch 2) builtin.

```
my $result = is_palindrome $x
# or in a pipeline:
@list |> map is_palindrome |> p
```

is_pangram

`is_pangram` — dsp / signal (extended) builtin.

```
my $result = is_pangram $x
# or in a pipeline:
@list |> map is_pangram |> p
```

is_pentagonal

`is_pentagonal` — number theory / primes builtin.

```
my $result = is_pentagonal $x
# or in a pipeline:
@list |> map is_pentagonal |> p
```

is_perfect

`is_perfect` — number theory / primes builtin.

```
my $result = is_perfect $x
# or in a pipeline:
@list |> map is_perfect |> p
```

is_perfect_square

`is_perfect_square` — predicates (batch 4) builtin.

```
my $result = is_perfect_square $x
# or in a pipeline:
@list |> map is_perfect_square |> p
```

is_permutation

`is_permutation` — predicates (batch 2) builtin.

```
my $result = is_permutation $x
# or in a pipeline:
@list |> map is_permutation |> p
```

is_phone_num

`is_phone_num` — extended stdlib builtin.

```
my $result = is_phone_num $x
# or in a pipeline:
@list |> map is_phone_num |> p
```

is_port_num

`is_port_num` — extended stdlib builtin.

```
my $result = is_port_num $x
# or in a pipeline:
@list |> map is_port_num |> p
```

is_positive

`is_positive` — predicates (batch 4) builtin.

```
my $result = is_positive $x
# or in a pipeline:
@list |> map is_positive |> p
```

is_positive_num

`is_positive_num` — extended stdlib builtin.

```
my $result = is_positive_num $x
# or in a pipeline:
@list |> map is_positive_num |> p
```

is_pow2

`is_pow2` — trivial numeric / predicate builtins builtin. Alias for `is_power_of_two`.

```
my $result = is_pow2 $x
# or in a pipeline:
@list |> map is_pow2 |> p
```

is_power_of

`is_power_of` — predicates (batch 4) builtin.

```
my $result = is_power_of $x
# or in a pipeline:
@list |> map is_power_of |> p
```

is_power_of_two

`is_power_of_two` — trivial numeric / predicate builtins builtin.

```
my $result = is_power_of_two $x
# or in a pipeline:
@list |> map is_power_of_two |> p
```

is_prefix

`is_prefix` — predicates (batch 4) builtin.

```
my $result = is_prefix $x
# or in a pipeline:
@list |> map is_prefix |> p
```

is_present

`is_present` — predicates (batch 4) builtin.

```
my $result = is_present $x
# or in a pipeline:
@list |> map is_present |> p
```

is_prime

`is_prime` — trivial numeric / predicate builtins builtin.

```
my $result = is_prime $x
# or in a pipeline:
@list |> map is_prime |> p
```

is_printable

`is_printable` — dsp / signal (extended) builtin.

```
my $result = is_printable $x
# or in a pipeline:
@list |> map is_printable |> p
```

is_probable_prime

`miller_rabin` (aliases `millerrabin`, `is_probable_prime`) performs a probabilistic primality test with k rounds (default 20).

```
p millerrabin(104729)     # 1 (prime)
p millerrabin(104730)     # 0 (composite)
```

is_readable

`is_readable` — file stat / path builtin.

```
my $result = is_readable $x
# or in a pipeline:
@list |> map is_readable |> p
```

is_ref

`is_ref` — trivial type predicates builtin.

```
my $result = is_ref $x
# or in a pipeline:
@list |> map is_ref |> p
```

is_regex_valid

`is_regex_valid` — more regex builtin.

```
my $result = is_regex_valid $x
# or in a pipeline:
@list |> map is_regex_valid |> p
```

is_root

`is_root` — more process / system builtin.

```
my $result = is_root $x
# or in a pipeline:
@list |> map is_root |> p
```

is_semver

`is_semver` — extended stdlib builtin.

```
my $result = is_semver $x
# or in a pipeline:
@list |> map is_semver |> p
```

is_slug

`is_slug` — extended stdlib builtin.

```
my $result = is_slug $x
# or in a pipeline:
@list |> map is_slug |> p
```

is_smith

`is_smith` — number theory / primes builtin.

```
my $result = is_smith $x
# or in a pipeline:
@list |> map is_smith |> p
```

is_sorted

`is_sorted` — predicates (batch 2) builtin.

```
my $result = is_sorted $x
# or in a pipeline:
@list |> map is_sorted |> p
```

is_sorted_by

`is_sorted_by` — additional missing stdlib functions builtin.

```
my $result = is_sorted_by $x
# or in a pipeline:
@list |> map is_sorted_by |> p
```

is_sorted_desc

`is_sorted_desc` — predicates (batch 2) builtin.

```
my $result = is_sorted_desc $x
# or in a pipeline:
@list |> map is_sorted_desc |> p
```

is_space

`is_space` — trivial string ops builtin.

```
my $result = is_space $x
# or in a pipeline:
@list |> map is_space |> p
```

is_sphenic

`is_sphenic` — batch 10 builtin. Alias for `sphenic_q`.

```
my $result = is_sphenic $x
# or in a pipeline:
@list |> map is_sphenic |> p
```

is_square

`is_square` — trivial numeric / predicate builtins builtin.

```
my $result = is_square $x
# or in a pipeline:
@list |> map is_square |> p
```

is_squarefree

`is_squarefree` — extended stdlib batch 3 builtin.

```
my $result = is_squarefree $x
# or in a pipeline:
@list |> map is_squarefree |> p
```

is_str

`is_str` — trivial type predicates builtin. Alias for `is_string`.

```
my $result = is_str $x
# or in a pipeline:
@list |> map is_str |> p
```

is_strictly_decreasing

`is_strictly_decreasing` — predicates (batch 4) builtin.

```
my $result = is_strictly_decreasing $x
# or in a pipeline:
@list |> map is_strictly_decreasing |> p
```

is_strictly_increasing

`is_strictly_increasing` — predicates (batch 4) builtin.

```
my $result = is_strictly_increasing $x
# or in a pipeline:
@list |> map is_strictly_increasing |> p
```

is_string

`is_string` — trivial type predicates builtin.

```
my $result = is_string $x
# or in a pipeline:
@list |> map is_string |> p
```

is_subset

`is_subset` — predicates (batch 2) builtin.

```
my $result = is_subset $x
# or in a pipeline:
@list |> map is_subset |> p
```

is_suffix

`is_suffix` — predicates (batch 4) builtin.

```
my $result = is_suffix $x
# or in a pipeline:
@list |> map is_suffix |> p
```

is_superset

is_superset — predicates (batch 2) builtin.

```
my $result = is_superset $x
# or in a pipeline:
@list |> map is_superset |> p
```

is_symlink

is_symlink — file stat / path builtin.

```
my $result = is_symlink $x
# or in a pipeline:
@list |> map is_symlink |> p
```

is_tree

is_tree — batch 24 builtin.

```
my $result = is_tree $x
# or in a pipeline:
@list |> map is_tree |> p
```

is_triangular

is_triangular — predicates (batch 4) builtin.

```
my $result = is_triangular $x
# or in a pipeline:
@list |> map is_triangular |> p
```

is_triple

is_triple — predicates (batch 2) builtin.

```
my $result = is_triple $x
# or in a pipeline:
@list |> map is_triple |> p
```

is_truthy

is_truthy — predicates (batch 4) builtin.

```
my $result = is_truthy $x
# or in a pipeline:
@list |> map is_truthy |> p
```

is_tty

is_tty — more process / system builtin.

```
my $result = is_tty $x
# or in a pipeline:
@list |> map is_tty |> p
```

is_undef

is_undef — trivial type predicates builtin.

```
my $result = is_undef $x
# or in a pipeline:
@list |> map is_undef |> p
```

is_upper

`is_upper` — trivial string ops builtin.

```
my $result = is_upper $x
# or in a pipeline:
@list |> map is_upper |> p
```

is_url

`is_url` — predicates (batch 4) builtin.

```
my $result = is_url $x
# or in a pipeline:
@list |> map is_url |> p
```

is_us_zip

`is_us_zip` — extended stdlib builtin.

```
my $result = is_us_zip $x
# or in a pipeline:
@list |> map is_us_zip |> p
```

is_uuid

`is_uuid` — id helpers builtin.

```
my $result = is_uuid $x
# or in a pipeline:
@list |> map is_uuid |> p
```

is_valid_cidr

`is_valid_cidr` — dsp / signal (extended) builtin.

```
my $result = is_valid_cidr $x
# or in a pipeline:
@list |> map is_valid_cidr |> p
```

is_valid_cron

`is_valid_cron` — dsp / signal (extended) builtin.

```
my $result = is_valid_cron $x
# or in a pipeline:
@list |> map is_valid_cron |> p
```

is_valid_date

`is_valid_date` — extended stdlib batch 3 builtin.

```
my $result = is_valid_date $x
# or in a pipeline:
@list |> map is_valid_date |> p
```

is_valid_email

`is_valid_email` — network / validation builtin.

```
my $result = is_valid_email $x
# or in a pipeline:
@list |> map is_valid_email |> p
```

is_valid_hex

`is_valid_hex` — file stat / path builtin.

```
my $result = is_valid_hex $x
# or in a pipeline:
@list |> map is_valid_hex |> p
```

is_valid_hex_color

`is_valid_hex_color` — dsp / signal (extended) builtin.

```
my $result = is_valid_hex_color $x
# or in a pipeline:
@list |> map is_valid_hex_color |> p
```

is_valid_ipv4

`is_valid_ipv4` — network / validation builtin.

```
my $result = is_valid_ipv4 $x
# or in a pipeline:
@list |> map is_valid_ipv4 |> p
```

is_valid_ipv6

`is_valid_ipv6` — network / validation builtin.

```
my $result = is_valid_ipv6 $x
# or in a pipeline:
@list |> map is_valid_ipv6 |> p
```

is_valid_latitude

`is_valid_latitude` — dsp / signal (extended) builtin.

```
my $result = is_valid_latitude $x
# or in a pipeline:
@list |> map is_valid_latitude |> p
```

is_valid_longitude

`is_valid_longitude` — dsp / signal (extended) builtin.

```
my $result = is_valid_longitude $x
# or in a pipeline:
@list |> map is_valid_longitude |> p
```

is_valid_mime

`is_valid_mime` — dsp / signal (extended) builtin.

```
my $result = is_valid_mime $x
# or in a pipeline:
@list |> map is_valid_mime |> p
```

is_valid_url

`is_valid_url` — network / validation builtin.

```
my $result = is_valid_url $x
# or in a pipeline:
@list |> map is_valid_url |> p
```

is_vowel

`is_vowel` — string (batch 2) builtin.

```
my $result = is_vowel $x
# or in a pipeline:
@list |> map is_vowel |> p
```

is_weekday

`is_weekday` — date (batch 2) builtin.

```
my $result = is_weekday $x
# or in a pipeline:
@list |> map is_weekday |> p
```

is_weekend

`is_weekend` — date (batch 2) builtin.

```
my $result = is_weekend $x
# or in a pipeline:
@list |> map is_weekend |> p
```

is_whitespace

`is_whitespace` — trivial string ops builtin. Alias for `is_space`.

```
my $result = is_whitespace $x
# or in a pipeline:
@list |> map is_whitespace |> p
```

is_whole

`is_whole` — predicates (batch 4) builtin.

```
my $result = is_whole $x
# or in a pipeline:
@list |> map is_whole |> p
```

is_whole_num

`is_whole_num` — extended stdlib builtin.

```
my $result = is_whole_num $x
# or in a pipeline:
@list |> map is_whole_num |> p
```

is_writable

`is_writable` — file stat / path builtin.

```
my $result = is_writable $x
# or in a pipeline:
@list |> map is_writable |> p
```

is_zero

`is_zero` — predicates (batch 4) builtin.

```
my $result = is_zero $x
# or in a pipeline:
@list |> map is_zero |> p
```

is_zero_num

`is_zero_num` — extended stdlib builtin.

```
my $result = is_zero_num $x
# or in a pipeline:
@list |> map is_zero_num |> p
```

isabun

`isabun` — number theory / primes builtin. Alias for `is_abundant`.

```
my $result = isabun $x
# or in a pipeline:
@list |> map isabun |> p
```

isanag

`isanag` — dsp / signal (extended) builtin. Alias for `is_anagram`.

```
my $result = isanag $x
# or in a pipeline:
@list |> map isanag |> p
```

isasc

`isasc` — extended stdlib builtin. Alias for `is_ascii`.

```
my $result = isasc $x
# or in a pipeline:
@list |> map isasc |> p
```

isb64

`isb64` — extended stdlib builtin. Alias for `is_base64`.

```
my $result = isb64 $x
# or in a pipeline:
@list |> map isb64 |> p
```

isbalp

`isbalp` — dsp / signal (extended) builtin. Alias for `is_balanced_parens`.

```
my $result = isbalp $x
# or in a pipeline:
@list |> map isbalp |> p
```

isbin

`isbin` — extended stdlib builtin. Alias for `is_binary_str`.

```
my $result = isbin $x
# or in a pipeline:
@list |> map isbin |> p
```

isbip

`isbip` — extended stdlib builtin. Alias for `is_bipartite_graph`.

```
my $result = isbip $x
# or in a pipeline:
@list |> map isbip |> p
```

isbn10

`isbn10` — extended stdlib builtin. Alias for `is_isbn10`.

```
my $result = isbn10 $x
# or in a pipeline:
@list |> map isbn10 |> p
```

isbn10_check

`isbn10_check` — batch 82 builtin.

```
my $result = isbn10_check $x
# or in a pipeline:
@list |> map isbn10_check |> p
```

isbn13

`isbn13` — extended stdlib builtin. Alias for `is_isbn13`.

```
my $result = isbn13 $x
# or in a pipeline:
@list |> map isbn13 |> p
```

isbn13_check

`isbn13_check` — batch 82 builtin.

```
my $result = isbn13_check $x
# or in a pipeline:
@list |> map isbn13_check |> p
```

iscc

`iscc` — extended stdlib builtin. Alias for `is_credit_card`.

```
my $result = iscc $x
# or in a pipeline:
@list |> map iscc |> p
```

iscidr

`iscidr` — dsp / signal (extended) builtin. Alias for `is_valid_cidr`.

```
my $result = iscidr $x
# or in a pipeline:
@list |> map iscidr |> p
```

isco_radius_kerr_step

`isco_radius_kerr_step` — batch 39 builtin.

```
my $result = isco_radius_kerr_step $x
# or in a pipeline:
@list |> map isco_radius_kerr_step |> p
```

iscopr

`iscopr` — extended stdlib batch 3 builtin. Alias for `is_coprime`.

```
my $result = iscopr $x
# or in a pipeline:
@list |> map iscopr |> p
```

iscron

`iscron` — dsp / signal (extended) builtin. Alias for `is_valid_cron`.

```
my $result = iscron $x
# or in a pipeline:
@list |> map iscron |> p
```

isctrl

`isctrl` — dsp / signal (extended) builtin. Alias for `is_control`.

```
my $result = isctrl $x
# or in a pipeline:
@list |> map isctrl |> p
```

isdefi

`isdefi` — number theory / primes builtin. Alias for `is_deficient`.

```
my $result = isdefi $x
# or in a pipeline:
@list |> map isdefi |> p
```

iseven

`iseven` — extended stdlib builtin. Alias for `is_even_num`.

```
my $result = iseven $x
# or in a pipeline:
@list |> map iseven |> p
```

isharsh

`isharsh` — extended stdlib batch 3 builtin. Alias for `is_harshad`.

```
my $result = isharsh $x
# or in a pipeline:
@list |> map isharsh |> p
```

ishet

`ishet` — extended stdlib batch 3 builtin. Alias for `is_heterogram`.

```
my $result = ishet $x
# or in a pipeline:
@list |> map ishet |> p
```

ishex

`ishex` — extended stdlib builtin. Alias for `is_hex_str`.

```
my $result = ishex $x
# or in a pipeline:
@list |> map ishex |> p
```

ishexc

`ishexc` — dsp / signal (extended) builtin. Alias for `is_valid_hex_color`.

```
my $result = ishexc $x
# or in a pipeline:
@list |> map ishexc |> p
```

ishost

`ishost` — extended stdlib builtin. Alias for `is_hostname_valid`.

```
my $result = ishost $x
# or in a pipeline:
@list |> map ishost |> p
```

isiban

`isiban` — extended stdlib builtin. Alias for `is_iban`.

```
my $result = isiban $x
# or in a pipeline:
@list |> map isiban |> p
```

isip4

`isip4` — extended stdlib builtin. Alias for `is_ipv4_addr`.

```
my $result = isip4 $x
# or in a pipeline:
@list |> map isip4 |> p
```

isip6

`isip6` — extended stdlib builtin. Alias for `is_ipv6_addr`.

```
my $result = isip6 $x
# or in a pipeline:
@list |> map isip6 |> p
```

isiso

`isiso` — extended stdlib batch 3 builtin. Alias for `is_isogram`.

```
my $result = isiso $x
# or in a pipeline:
@list |> map isiso |> p
```

isisodt

`isisodt` — extended stdlib builtin. Alias for `is_iso_date`.

```
my $result = isisodt $x
# or in a pipeline:
@list |> map isisodt |> p
```

isisodtm

`isisodtm` — extended stdlib builtin. Alias for `is_iso_datetime`.

```
my $result = isisodtm $x
# or in a pipeline:
@list |> map isisodtm |> p
```

isisotm

`isisotm` — extended stdlib builtin. Alias for `is_iso_time`.

```
my $result = isisotm $x
# or in a pipeline:
@list |> map isisotm |> p
```

isjson

`isjson` — extended stdlib builtin. Alias for `is_json`.

```
my $result = isjson $x
# or in a pipeline:
@list |> map isjson |> p
```

iskap

`iskap` — extended stdlib batch 3 builtin. Alias for `is_kaprekar`.

```
my $result = iskap $x
# or in a pipeline:
@list |> map iskap |> p
```

isl

`isl` — python/ruby stdlib builtin. Alias for `islice`.

```
my $result = isl $x
# or in a pipeline:
@list |> map isl |> p
```

islamic_leap_year

`islamic_leap_year` — batch 52 builtin.

```
my $result = islamic_leap_year $x
# or in a pipeline:
@list |> map islamic_leap_year |> p
```

island_equilibrium

`island_equilibrium` — batch 22 builtin.

```
my $result = island_equilibrium $x
# or in a pipeline:
@list |> map island_equilibrium |> p
```

islat

`islat` — dsp / signal (extended) builtin. Alias for `is_valid_latitude`.

```
my $result = islat $x
# or in a pipeline:
@list |> map islat |> p
```

islice

`islice` — python/ruby stdlib builtin.

```
my $result = islice $x
# or in a pipeline:
@list |> map islice |> p
```

islon

`islon` — dsp / signal (extended) builtin. Alias for `is_valid_longitude`.

```
my $result = islon $x
# or in a pipeline:
@list |> map islon |> p
```

ism

`ism` — extended stdlib builtin. Alias for `is_match`.

```
my $result = ism $x
# or in a pipeline:
@list |> map ism |> p
```

ismac

`ismac` — extended stdlib builtin. Alias for `is_mac_addr`.

```
my $result = ismac $x
# or in a pipeline:
@list |> map ismac |> p
```

ismime

`ismime` — dsp / signal (extended) builtin. Alias for `is_valid_mime`.

```
my $result = ismime $x
# or in a pipeline:
@list |> map ismime |> p
```

ismono

`ismono` — extended stdlib batch 3 builtin. Alias for `is_monotonic`.

```
my $result = ismono $x
# or in a pipeline:
@list |> map ismono |> p
```

isnarc

`isnarc` — extended stdlib batch 3 builtin. Alias for `is_narcissistic`.

```
my $result = isnarc $x
# or in a pipeline:
@list |> map isnarc |> p
```

isneg

`isneg` — extended stdlib builtin. Alias for `is_negative_num`.

```
my $result = isneg $x
# or in a pipeline:
@list |> map isneg |> p
```

isnumstr

`isnumstr` — dsp / signal (extended) builtin. Alias for `is_numeric_string`.

```
my $result = isnumstr $x
# or in a pipeline:
@list |> map isnumstr |> p
```

iso226_phon_adjustment

`iso226_phon_adjustment` — batch 30 builtin.

```
my $result = iso226_phon_adjustment $x
# or in a pipeline:
@list |> map iso226_phon_adjustment |> p
```

iso8601_duration_parse

`iso8601_duration_parse` — batch 82 builtin.

```
my $result = iso8601_duration_parse $x
# or in a pipeline:
@list |> map iso8601_duration_parse |> p
```

iso8601_duration_to_seconds

`iso8601_duration_to_seconds` — batch 82 builtin.

```
my $result = iso8601_duration_to_seconds $input
```

iso_day_number

`iso_day_number` — batch 52 builtin.

```
my $result = iso_day_number $x
# or in a pipeline:
@list |> map iso_day_number |> p
```

iso_dow

`iso_dow` — batch 9 builtin. Alias for `day_of_week_iso`.

```
my $result = iso_dow $x
# or in a pipeline:
@list |> map iso_dow |> p
```

iso_ordinal_date

`iso_ordinal_date` — batch 10 builtin.

```
my $result = iso_ordinal_date $x
# or in a pipeline:
@list |> map iso_ordinal_date |> p
```

iso_week

`iso_week` — batch 9 builtin. Alias for `iso_week_number`.

```
my $result = iso_week $x
# or in a pipeline:
@list |> map iso_week |> p
```

iso_week_date

`iso_week_date` — batch 52 builtin.

```
my $result = iso_week_date $x
# or in a pipeline:
@list |> map iso_week_date |> p
```

iso_week_number

`iso_week_number` — batch 9 builtin.

```
my $result = iso_week_number $x
# or in a pipeline:
@list |> map iso_week_number |> p
```

isoct

`isoct` — extended stdlib builtin. Alias for `is_octal_str`.

```
my $result = isoct $x
# or in a pipeline:
@list |> map isoct |> p
```

isodd

`isodd` — extended stdlib builtin. Alias for `is_odd_num`.

```
my $result = isodd $x
# or in a pipeline:
@list |> map isodd |> p
```

isoelectric_point_protein

`isoelectric_point_protein` — batch 63 builtin.

```
my $result = isoelectric_point_protein $x
# or in a pipeline:
@list |> map isoelectric_point_protein |> p
```

isolated_horizon_charge

`isolated_horizon_charge` — batch 39 builtin.

```
my $result = isolated_horizon_charge $x
# or in a pipeline:
@list |> map isolated_horizon_charge |> p
```

isotropic_relation_check

`isotropic_relation_check` — batch 42 builtin.

```
my $result = isotropic_relation_check $x
# or in a pipeline:
@list |> map isotropic_relation_check |> p
```

isp

`isp` — haskell list functions builtin. Alias for `intersperse_val`.

```
my $result = isp $x
# or in a pipeline:
@list |> map isp |> p
```

ispang

`ispang` — dsp / signal (extended) builtin. Alias for `is_pangram`.

```
my $result = ispang $x
# or in a pipeline:
@list |> map ispang |> p
```

ispent

`ispent` — number theory / primes builtin. Alias for `is_pentagonal`.

```
my $result = ispent $x
# or in a pipeline:
@list |> map ispent |> p
```

isperf

`isperf` — number theory / primes builtin. Alias for `is_perfect`.

```
my $result = isperf $x
# or in a pipeline:
@list |> map isperf |> p
```

isphone

`isphone` — extended stdlib builtin. Alias for `is_phone_num`.

```
my $result = isphone $x
# or in a pipeline:
@list |> map isphone |> p
```

isport

`isport` — extended stdlib builtin. Alias for `is_port_num`.

```
my $result = isport $x
# or in a pipeline:
@list |> map isport |> p
```

ispos

`ispos` — extended stdlib builtin. Alias for `is_positive_num`.

```
my $result = ispos $x
# or in a pipeline:
@list |> map ispos |> p
```

isprint

`isprint` — dsp / signal (extended) builtin. Alias for `is_printable`.

```
my $result = isprint $x
# or in a pipeline:
@list |> map isprint |> p
```

ispw

`ispw` — additional missing stdlib functions builtin. Alias for `intersperse_with`.

```
my $result = ispw $x
# or in a pipeline:
@list |> map ispw |> p
```

isslug

`isslug` — extended stdlib builtin. Alias for `is_slug`.

```
my $result = isslug $x
# or in a pipeline:
@list |> map isslug |> p
```

issmith

`issmith` — number theory / primes builtin. Alias for `is_smith`.

```
my $result = issmith $x
# or in a pipeline:
@list |> map issmith |> p
```

issqfr

`issqfr` — extended stdlib batch 3 builtin. Alias for `is_squarefree`.

```
my $result = issqfr $x
# or in a pipeline:
@list |> map issqfr |> p
```

issrtb

`issrtb` — additional missing stdlib functions builtin. Alias for `is_sorted_by`.

```
my $result = issrtb $x
# or in a pipeline:
@list |> map issrtb |> p
```

issv

`issv` — extended stdlib builtin. Alias for `is_semver`.

```
my $result = issv $x
# or in a pipeline:
@list |> map issv |> p
```

ista_step

`ista_step` — batch 41 builtin.

```
my $result = ista_step $x
# or in a pipeline:
@list |> map ista_step |> p
```

istft_step

`istft_step` — batch 74 builtin.

```
my $result = istft_step $x
# or in a pipeline:
@list |> map istft_step |> p
```

isvdate

`isvdate` — extended stdlib batch 3 builtin. Alias for `is_valid_date`.

```
my $result = isvdate $x
# or in a pipeline:
@list |> map isvdate |> p
```

iswap_gate

`iswap_gate` — batches 14-16 builtin.

```
my $result = iswap_gate $x
# or in a pipeline:
@list |> map iswap_gate |> p
```

isweak

`isweak` — list / aggregate builtin.

```
my $result = isweak $x
# or in a pipeline:
@list |> map isweak |> p
```

iswhole

`iswhole` — extended stdlib builtin. Alias for `is_whole_num`.

```
my $result = iswhole $x
# or in a pipeline:
@list |> map iswhole |> p
```

iszero

`iszero` — extended stdlib builtin. Alias for `is_zero_num`.

```
my $result = iszero $x
# or in a pipeline:
@list |> map iszero |> p
```

iszip

`iszip` — extended stdlib builtin. Alias for `is_us_zip`.

```
my $result = iszip $x
# or in a pipeline:
@list |> map iszip |> p
```

italic

`italic` — color / ansi builtin. Alias for `red`.

```
my $result = italic $x
# or in a pipeline:
@list |> map italic |> p
```

itcz_position_lat

`itcz_position_lat` — batch 42 builtin.

```
my $result = itcz_position_lat $x
# or in a pipeline:
@list |> map itcz_position_lat |> p
```

iter

`iter` — algebraic match builtin. Alias for `iterate`.

```
my $result = iter $x
# or in a pipeline:
@list |> map iter |> p
```

iterate

`iterate` — algebraic match builtin.

```
my $result = iterate $x
# or in a pipeline:
@list |> map iterate |> p
```

iterate_n

`iterate_n` — functional combinators builtin.

```
my $result = iterate_n $x
# or in a pipeline:
@list |> map iterate_n |> p
```

iv

`iv` — batch 72 builtin.

```
my $result = iv $x
# or in a pipeline:
@list |> map iv |> p
```

iv_estimator

`iv_estimator` — batch 36 builtin.

```
my $result = iv_estimator $x
# or in a pipeline:
@list |> map iv_estimator |> p
```

jaccard

`jaccard` — extended stdlib batch 3 builtin. Alias for `jaccard_index`.

```
my $result = jaccard $x
# or in a pipeline:
@list |> map jaccard |> p
```

jaccard_index

`jaccard_index` — extended stdlib batch 3 builtin.

```
my $result = jaccard_index $x
# or in a pipeline:
@list |> map jaccard_index |> p
```

jaccard_similarity

`jaccard_similarity` — math functions builtin.

```
my $result = jaccard_similarity $x
# or in a pipeline:
@list |> map jaccard_similarity |> p
```

jacobi_cn_small_q

`jacobi_cn_small_q` — batch 29 builtin.

```
my $result = jacobi_cn_small_q $x
# or in a pipeline:
@list |> map jacobi_cn_small_q |> p
```

jacobi_dn_small_q

`jacobi_dn_small_q` — batch 29 builtin.

```
my $result = jacobi_dn_small_q $x
# or in a pipeline:
@list |> map jacobi_dn_small_q |> p
```

jacobi_sn_small_q

`jacobi_sn_small_q` — batch 29 builtin.

```
my $result = jacobi_sn_small_q $x
# or in a pipeline:
@list |> map jacobi_sn_small_q |> p
```

jacobian_2dof

`jacobian_2dof` — batch 59 builtin.

```
my $result = jacobian_2dof $x
# or in a pipeline:
@list |> map jacobian_2dof |> p
```

jahn_teller_check

`jahn_teller_check` — batch 63 builtin.

```
my $result = jahn_teller_check $x
# or in a pipeline:
@list |> map jahn_teller_check |> p
```

jaro_similarity

`jaro_similarity` — extended stdlib batch 3 builtin.

```
my $result = jaro_similarity $x
# or in a pipeline:
@list |> map jaro_similarity |> p
```

jarosim

`jarosim` — extended stdlib batch 3 builtin. Alias for `jaro_similarity`.

```
my $result = jarosim $x
# or in a pipeline:
@list |> map jarosim |> p
```

jd

`jd` — data / network builtin. Alias for `json_decode`.

```
my $result = jd $x
# or in a pipeline:
@list |> map jd |> p
```

jd_to_calendar

`jd_to_calendar` — batch 55 builtin.

```
my $result = jd_to_calendar $input
```

je

`je` — data / network builtin. Alias for `json_encode`.

```
my $result = je $x
# or in a pipeline:
@list |> map je |> p
```

jeans_length

`jeans_length` — batch 12 builtin.

```
my $result = jeans_length $x
# or in a pipeline:
@list |> map jeans_length |> p
```

jenkins_oat

`jenkins_oat` — batch 11 builtin. Alias for `jenkins_one_at_a_time`.

```
my $result = jenkins_oat $x
# or in a pipeline:
@list |> map jenkins_oat |> p
```

jenkins_one_at_a_time

`jenkins_one_at_a_time` — batch 11 builtin.

```
my $result = jenkins_one_at_a_time $x
# or in a pipeline:
@list |> map jenkins_one_at_a_time |> p
```

jensen_alpha

`jensen_alpha` — batch 20 builtin.

```
my $result = jensen_alpha $x
# or in a pipeline:
@list |> map jensen_alpha |> p
```

jensen_shannon_div

`jensen_shannon_div` — batch 40 builtin.

```
my $result = jensen_shannon_div $x
# or in a pipeline:
@list |> map jensen_shannon_div |> p
```

jensens_alpha

`jensens_alpha` — batch 13 builtin.

```
my $result = jensens_alpha $x
# or in a pipeline:
@list |> map jensens_alpha |> p
```

job_shop_makespan_lower

`job_shop_makespan_lower` — batch 41 builtin.

```
my $result = job_shop_makespan_lower $x
# or in a pipeline:
@list |> map job_shop_makespan_lower |> p
```

johansen_test

`johansen_test` — batch 78 builtin.

```
my $result = johansen_test $x
# or in a pipeline:
@list |> map johansen_test |> p
```

johansen_trace_step

`johansen_trace_step` — batch 36 builtin.

```
my $result = johansen_trace_step $x
# or in a pipeline:
@list |> map johansen_trace_step |> p
```

johnson_2m

`johnson_2m` — batch 10 builtin. Alias for `johnsons_rule`.

```
my $result = johnson_2m $x
# or in a pipeline:
@list |> map johnson_2m |> p
```

johnson_reweight

`johnson_reweight` — batch 75 builtin.

```
my $result = johnson_reweight $x
# or in a pipeline:
@list |> map johnson_reweight |> p
```

johnsons_rule

`johnsons_rule` — batch 10 builtin.

```
my $result = johnsons_rule $x
# or in a pipeline:
@list |> map johnsons_rule |> p
```

join_colons

`join_colons` — conversion / utility (batch 4) builtin.

```
my $result = join_colons $x
# or in a pipeline:
@list |> map join_colons |> p
```

join_commas

`join_commas` — conversion / utility (batch 4) builtin.

```
my $result = join_commas $x
# or in a pipeline:
@list |> map join_commas |> p
```

join_dashes

`join_dashes` — conversion / utility (batch 4) builtin.

```
my $result = join_dashes $x
# or in a pipeline:
@list |> map join_dashes |> p
```

join_dots

`join_dots` — conversion / utility (batch 4) builtin.

```
my $result = join_dots $x
# or in a pipeline:
@list |> map join_dots |> p
```

join_lines

`join_lines` — conversion / utility (batch 4) builtin.

```
my $result = join_lines $x
# or in a pipeline:
@list |> map join_lines |> p
```

join_pipes

`join_pipes` — conversion / utility (batch 4) builtin.

```
my $result = join_pipes $x
# or in a pipeline:
@list |> map join_pipes |> p
```

join_slashes

`join_slashes` — conversion / utility (batch 4) builtin.

```
my $result = join_slashes $x
# or in a pipeline:
@list |> map join_slashes |> p
```

join_spaces

`join_spaces` — conversion / utility (batch 4) builtin.

```
my $result = join_spaces $x
# or in a pipeline:
@list |> map join_spaces |> p
```

join_tabs

`join_tabs` — conversion / utility (batch 4) builtin.

```
my $result = join_tabs $x
# or in a pipeline:
@list |> map join_tabs |> p
```

joint_entropy_step

`joint_entropy_step` — batch 40 builtin.

```
my $result = joint_entropy_step $x
# or in a pipeline:
@list |> map joint_entropy_step |> p
```

jones_polynomial_at_i

`jones_polynomial_at_i` — batch 37 builtin.

```
my $result = jones_polynomial_at_i $x
# or in a pipeline:
@list |> map jones_polynomial_at_i |> p
```

jones_polynomial_at_minus_one

`jones_polynomial_at_minus_one` — batch 37 builtin.

```
my $result = jones_polynomial_at_minus_one $x
# or in a pipeline:
@list |> map jones_polynomial_at_minus_one |> p
```

joule_heating_battery

`joule_heating_battery` — batch 38 builtin.

```
my $result = joule_heating_battery $x
# or in a pipeline:
@list |> map joule_heating_battery |> p
```

joules_to_cal

joules_to_cal — file stat / path builtin.

```
my $result = joules_to_cal $input
```

joules_to_ev

joules_to_ev — batch 30 builtin.

```
my $result = joules_to_ev $input
```

jpeg_dct_8x8_quant

jpeg_dct_8x8_quant — batch 40 builtin.

```
my $result = jpeg_dct_8x8_quant $x
# or in a pipeline:
@list |> map jpeg_dct_8x8_quant |> p
```

jpeg_zig_zag_index

jpeg_zig_zag_index — batch 40 builtin.

```
my $result = jpeg_zig_zag_index $x
# or in a pipeline:
@list |> map jpeg_zig_zag_index |> p
```

jr_european_call

jr_european_call — batch 20 builtin.

```
my $result = jr_european_call $x
# or in a pipeline:
@list |> map jr_european_call |> p
```

json_array_length

json_array_length — batch 70 builtin.

```
my $result = json_array_length $x
# or in a pipeline:
@list |> map json_array_length |> p
```

json_build_object

json_build_object — batch 70 builtin.

```
my $result = json_build_object $x
# or in a pipeline:
@list |> map json_build_object |> p
```

json_each

json_each — batch 70 builtin.

```
my $result = json_each $x
# or in a pipeline:
@list |> map json_each |> p
```

json_escape

`json_escape` — json helpers builtin. Alias for `escape_json`.

```
my $result = json_escape $x
# or in a pipeline:
@list |> map json_escape |> p
```

json_extract_path

`json_extract_path` — batch 70 builtin.

```
my $result = json_extract_path $x
# or in a pipeline:
@list |> map json_extract_path |> p
```

json_minify

`json_minify` — json helpers builtin.

```
my $result = json_minify $x
# or in a pipeline:
@list |> map json_minify |> p
```

json_pretty

`json_pretty` — json helpers builtin.

```
my $result = json_pretty $x
# or in a pipeline:
@list |> map json_pretty |> p
```

json_strip_nulls

`json_strip_nulls` — batch 70 builtin.

```
my $result = json_strip_nulls $x
# or in a pipeline:
@list |> map json_strip_nulls |> p
```

jsonb_array_length

`jsonb_array_length` — batch 70 builtin.

```
my $result = jsonb_array_length $x
# or in a pipeline:
@list |> map jsonb_array_length |> p
```

jsonb_object_keys

`jsonb_object_keys` — batch 70 builtin.

```
my $result = jsonb_object_keys $x
# or in a pipeline:
@list |> map jsonb_object_keys |> p
```

jsonb_path_query

`jsonb_path_query` — batch 70 builtin.

```
my $result = jsonb_path_query $x
# or in a pipeline:
@list |> map jsonb_path_query |> p
```

jsonb_pretty

`jsonb_pretty` — batch 70 builtin.

```
my $result = jsonb_pretty $x
# or in a pipeline:
@list |> map jsonb_pretty |> p
```

jsonb_set

`jsonb_set` — batch 70 builtin.

```
my $result = jsonb_set $x
# or in a pipeline:
@list |> map jsonb_set |> p
```

jsonb_typeof

`jsonb_typeof` — batch 70 builtin.

```
my $result = jsonb_typeof $x
# or in a pipeline:
@list |> map jsonb_typeof |> p
```

jukes_cantor_distance

`jukes_cantor_distance` — batch 33 builtin.

```
my $result = jukes_cantor_distance $x
# or in a pipeline:
@list |> map jukes_cantor_distance |> p
```

julian_centuries_j2000

`julian_centuries_j2000` — batches 14-16 builtin.

```
my $result = julian_centuries_j2000 $x
# or in a pipeline:
@list |> map julian_centuries_j2000 |> p
```

julian_day

`julian_day` — batch 55 builtin.

```
my $result = julian_day $x
# or in a pipeline:
@list |> map julian_day |> p
```

julian_from_fixed

`julian_from_fixed` — batch 52 builtin.

```
my $result = julian_from_fixed $x
# or in a pipeline:
@list |> map julian_from_fixed |> p
```

just_intonation_freq

`just_intonation_freq` — batch 64 builtin.

```
my $result = just_intonation_freq $x
# or in a pipeline:
@list |> map just_intonation_freq |> p
```

just_intonation_ratio

`just_intonation_ratio` — batch 30 builtin.

```
my $result = just_intonation_ratio $x
# or in a pipeline:
@list |> map just_intonation_ratio |> p
```

juxt2

`juxt2` — functional combinators builtin.

```
my $result = juxt2 $x
# or in a pipeline:
@list |> map juxt2 |> p
```

juxt3

`juxt3` — functional combinators builtin.

```
my $result = juxt3 $x
# or in a pipeline:
@list |> map juxt3 |> p
```

jv

`jv` — batch 72 builtin.

```
my $result = jv $x
# or in a pipeline:
@list |> map jv |> p
```

k_coreness

`k_coreness` — batch 24 builtin.

```
my $result = k_coreness $x
# or in a pipeline:
@list |> map k_coreness |> p
```

k_from_dg

`k_from_dg` — batch 21 builtin.

```
my $result = k_from_dg $x
# or in a pipeline:
@list |> map k_from_dg |> p
```

k_shortest_spanning

`k_shortest_spanning` — batch 75 builtin.

```
my $result = k_shortest_spanning $x
# or in a pipeline:
@list |> map k_shortest_spanning |> p
```

k_to_c

k_to_c — unit conversions builtin.

```
my $result = k_to_c $input
```

k_to_f

k_to_f — unit conversions builtin.

```
my $result = k_to_f $input
```

ka_to_pka

ka_to_pka — batch 63 builtin.

```
my $result = ka_to_pka $input
```

kaiser

window_kaiser($n, $beta) (alias kaiser) — generates a Kaiser window with parameter beta controlling the tradeoff between main lobe width and sidelobe level. Beta=0 is rectangular, beta~5 is similar to Hamming.

```
my $w = kaiser(1024, 5.0)
```

kaiser_window

kaiser_window — batch 25 builtin.

```
my $result = kaiser_window $x
# or in a pipeline:
@list |> map kaiser_window |> p
```

kaiserord_step

kaiserord_step — batch 74 builtin.

```
my $result = kaiserord_step $x
# or in a pipeline:
@list |> map kaiserord_step |> p
```

kalai_smorodinsky_step

kalai_smorodinsky_step — batch 43 builtin.

```
my $result = kalai_smorodinsky_step $x
# or in a pipeline:
@list |> map kalai_smorodinsky_step |> p
```

kalman_filter_step

kalman_filter_step — batch 78 builtin.

```
my $result = kalman_filter_step $x
# or in a pipeline:
@list |> map kalman_filter_step |> p
```

kalman_predict_state

kalman_predict_state — batch 40 builtin.

```
my $result = kalman_predict_state $x
# or in a pipeline:
@list |> map kalman_predict_state |> p
```

kalman_smoother_step

`kalman_smoother_step` — batch 78 builtin.

```
my $result = kalman_smoother_step $x
# or in a pipeline:
@list |> map kalman_smoother_step |> p
```

kalman_update_state

`kalman_update_state` — batch 40 builtin.

```
my $result = kalman_update_state $x
# or in a pipeline:
@list |> map kalman_update_state |> p
```

kaluza_klein_step

`kaluza_klein_step` — batch 39 builtin.

```
my $result = kaluza_klein_step $x
# or in a pipeline:
@list |> map kaluza_klein_step |> p
```

kaprekar_routine_step

`kaprekar_routine_step` — batch 11 builtin.

```
my $result = kaprekar_routine_step $x
# or in a pipeline:
@list |> map kaprekar_routine_step |> p
```

karger_contract_edge

`karger_contract_edge` — batch 41 builtin.

```
my $result = karger_contract_edge $x
# or in a pipeline:
@list |> map karger_contract_edge |> p
```

karger_min_cut_count

`karger_min_cut_count` — batch 41 builtin.

```
my $result = karger_min_cut_count $x
# or in a pipeline:
@list |> map karger_min_cut_count |> p
```

karger_step

`karger_step` — batch 75 builtin.

```
my $result = karger_step $x
# or in a pipeline:
@list |> map karger_step |> p
```

karhunen_loeve_step

karhunen_loeve_step — batch 40 builtin.

```
my $result = karhunen_loeve_step $x
# or in a pipeline:
@list |> map karhunen_loeve_step |> p
```

kasiski_repeats

kasiski_repeats — batch 26 builtin.

```
my $result = kasiski_repeats $x
# or in a pipeline:
@list |> map kasiski_repeats |> p
```

katz_centrality

katz_centrality — batch 24 builtin.

```
my $result = katz_centrality $x
# or in a pipeline:
@list |> map katz_centrality |> p
```

kauffman_bracket_eval

kauffman_bracket_eval — batch 37 builtin.

```
my $result = kauffman_bracket_eval $x
# or in a pipeline:
@list |> map kauffman_bracket_eval |> p
```

kb_to_b

kb_to_b — unit conversions builtin. Alias for kb_to_bytes.

```
my $result = kb_to_b $input
```

kb_to_bytes

kb_to_bytes — unit conversions builtin.

```
my $result = kb_to_bytes $input
```

kb_to_mb

kb_to_mb — unit conversions builtin.

```
my $result = kb_to_mb $input
```

kc_from_rates

kc_from_rates — batch 21 builtin.

```
my $result = kc_from_rates $x
# or in a pipeline:
@list |> map kc_from_rates |> p
```

kcoulomb

`kcoulomb` — physics constants builtin. Alias for `coulomb_constant`.

```
my $result = kcoulomb $x
# or in a pipeline:
@list |> map kcoulomb |> p
```

kde_bandwidth_lscv

`kde_bandwidth_lscv` — batch 82 builtin.

```
my $result = kde_bandwidth_lscv $x
# or in a pipeline:
@list |> map kde_bandwidth_lscv |> p
```

kde_biweight

`kde_biweight` — batch 82 builtin.

```
my $result = kde_biweight $x
# or in a pipeline:
@list |> map kde_biweight |> p
```

kde_cosine

`kde_cosine` — batch 82 builtin.

```
my $result = kde_cosine $x
# or in a pipeline:
@list |> map kde_cosine |> p
```

kde_epanechnikov

`kde_epanechnikov` — batch 82 builtin.

```
my $result = kde_epanechnikov $x
# or in a pipeline:
@list |> map kde_epanechnikov |> p
```

kde_gaussian_2d

`kde_gaussian_2d` — batch 82 builtin.

```
my $result = kde_gaussian_2d $x
# or in a pipeline:
@list |> map kde_gaussian_2d |> p
```

kde_logistic_kernel

`kde_logistic_kernel` — batch 82 builtin.

```
my $result = kde_logistic_kernel $x
# or in a pipeline:
@list |> map kde_logistic_kernel |> p
```

kde_scott_bw

`kde_scott_bw` — batch 82 builtin.

```
my $result = kde_scott_bw $x
# or in a pipeline:
@list |> map kde_scott_bw |> p
```

kde_silverman_bw

`kde_silverman_bw` — batch 82 builtin.

```
my $result = kde_silverman_bw $x
# or in a pipeline:
@list |> map kde_silverman_bw |> p
```

kde_triangular

`kde_triangular` — batch 82 builtin.

```
my $result = kde_triangular $x
# or in a pipeline:
@list |> map kde_triangular |> p
```

kde_triweight

`kde_triweight` — batch 82 builtin.

```
my $result = kde_triweight $x
# or in a pipeline:
@list |> map kde_triweight |> p
```

kde_uniform

`kde_uniform` — batch 82 builtin.

```
my $result = kde_uniform $x
# or in a pipeline:
@list |> map kde_uniform |> p
```

keep

`keep` — algebraic match builtin.

```
my $result = keep $x
# or in a pipeline:
@list |> map keep |> p
```

keep_if

`keep_if` — functional combinators builtin.

```
my $result = keep_if $x
# or in a pipeline:
@list |> map keep_if |> p
```

keith_q

`keith_q` — batch 11 builtin.

```
my $result = keith_q $x
# or in a pipeline:
@list |> map keith_q |> p
```

kelly_criterion

kelly_criterion — batches 17-18 builtin.

```
my $result = kelly_criterion $x
# or in a pipeline:
@list |> map kelly_criterion |> p
```

kemeny_score_step

kemeny_score_step — batch 43 builtin.

```
my $result = kemeny_score_step $x
# or in a pipeline:
@list |> map kemeny_score_step |> p
```

kendall

kendall_tau (aliases kendall, ktau) computes Kendall's rank correlation coefficient (tau-b).

```
p ktau([1,2,3,4], [1,2,4,3])  # 0.67
```

kendalltau

kendalltau — batch 53 builtin.

```
my $result = kendalltau $x
# or in a pipeline:
@list |> map kendalltau |> p
```

kepler_hyperbolic

kepler_hyperbolic — batches 14-16 builtin.

```
my $result = kepler_hyperbolic $x
# or in a pipeline:
@list |> map kepler_hyperbolic |> p
```

kepler_period_au

kepler_period_au — batch 12 builtin.

```
my $result = kepler_period_au $x
# or in a pipeline:
@list |> map kepler_period_au |> p
```

kerr_ergosphere_eq

kerr_ergosphere_eq — batch 31 builtin.

```
my $result = kerr_ergosphere_eq $x
# or in a pipeline:
@list |> map kerr_ergosphere_eq |> p
```

kerr_horizon

kerr_horizon — batch 31 builtin.

```
my $result = kerr_horizon $x
# or in a pipeline:
@list |> map kerr_horizon |> p
```

kerr_newman_charge_term

kerr_newman_charge_term — batch 39 builtin.

```
my $result = kerr_newman_charge_term $x
# or in a pipeline:
@list |> map kerr_newman_charge_term |> p
```

key_signature_flats

key_signature_flats — batch 64 builtin.

```
my $result = key_signature_flats $x
# or in a pipeline:
@list |> map key_signature_flats |> p
```

key_signature_for

key_signature_for — batch 9 builtin.

```
my $result = key_signature_for $x
# or in a pipeline:
@list |> map key_signature_for |> p
```

key_signature_sharps

key_signature_sharps — batch 64 builtin.

```
my $result = key_signature_sharps $x
# or in a pipeline:
@list |> map key_signature_sharps |> p
```

keys_sorted

keys_sorted — hash ops (batch 2) builtin.

```
my $result = keys_sorted $x
# or in a pipeline:
@list |> map keys_sorted |> p
```

kg_to_lb

kg_to_lb — batch 30 builtin.

```
my $result = kg_to_lb $input
```

kg_to_lbs

kg_to_lbs — unit conversions builtin.

```
my $result = kg_to_lbs $input
```

kg_to_stone

kg_to_stone — unit conversions builtin.

```
my $result = kg_to_stone $input
```

khinchin

`khinchin` — math constants builtin. Alias for `khinchin_constant`.

```
my $result = khinchin $x
# or in a pipeline:
@list |> map khinchin |> p
```

khinchin_constant

`khinchin_constant` — math constants builtin.

```
my $result = khinchin_constant $x
# or in a pipeline:
@list |> map khinchin_constant |> p
```

khovanov_q_grading

`khovanov_q_grading` — batch 37 builtin.

```
my $result = khovanov_q_grading $x
# or in a pipeline:
@list |> map khovanov_q_grading |> p
```

khovanov_rasmussen_s

`khovanov_rasmussen_s` — batch 37 builtin.

```
my $result = khovanov_rasmussen_s $x
# or in a pipeline:
@list |> map khovanov_rasmussen_s |> p
```

killing_form_su2

`killing_form_su2` — batch 37 builtin.

```
my $result = killing_form_su2 $x
# or in a pipeline:
@list |> map killing_form_su2 |> p
```

killing_vector_lie_step

`killing_vector_lie_step` — batch 39 builtin.

```
my $result = killing_vector_lie_step $x
# or in a pipeline:
@list |> map killing_vector_lie_step |> p
```

kimura_2p_distance

`kimura_2p_distance` — batch 33 builtin.

```
my $result = kimura_2p_distance $x
# or in a pipeline:
@list |> map kimura_2p_distance |> p
```

kinetic_energy

`kinetic_energy` — physics formulas builtin.

```
my $result = kinetic_energy $x
# or in a pipeline:
@list |> map kinetic_energy |> p
```

kirnberger_iii

kirnberger_iii — batch 64 builtin.

```
my $result = kirnberger_iii $x
# or in a pipeline:
@list |> map kirnberger_iii |> p
```

kleibers_law

kleibers_law — batch 22 builtin.

```
my $result = kleibers_law $x
# or in a pipeline:
@list |> map kleibers_law |> p
```

km_to_miles

km_to_miles — unit conversions builtin.

```
my $result = km_to_miles $input
```

kmeans

kmeans — k-means clustering (Lloyd's algorithm). Takes array of points and k. Returns cluster assignments. Like R's kmeans().

```
my @clusters = @{kmeans([[0,0],[1,0],[10,10],[11,10]], 2)}
# [0,0,1,1] - two clusters
```

kmer_jaccard

kmer_jaccard — batch 33 builtin.

```
my $result = kmer_jaccard $x
# or in a pipeline:
@list |> map kmer_jaccard |> p
```

kmh_to_mph

kmh_to_mph — batch 30 builtin.

```
my $result = kmh_to_mph $input
```

kmh_to_mps

kmh_to_mps — batch 30 builtin.

```
my $result = kmh_to_mps $input
```

kmp

kmp — batch 10 builtin. Alias for kmp_failure.

```
my $result = kmp $x
# or in a pipeline:
@list |> map kmp |> p
```

kmp_failure

kmp_failure — batch 10 builtin.

```
my $result = kmp_failure $x
# or in a pipeline:
@list |> map kmp_failure |> p
```

knapsack_01_dp_value

knapsack_01_dp_value — batch 41 builtin.

```
my $result = knapsack_01_dp_value $x
# or in a pipeline:
@list |> map knapsack_01_dp_value |> p
```

knapsack_branch_bound

knapsack_branch_bound — batch 41 builtin.

```
my $result = knapsack_branch_bound $x
# or in a pipeline:
@list |> map knapsack_branch_bound |> p
```

knapsack_fractional_step

knapsack_fractional_step — batch 41 builtin.

```
my $result = knapsack_fractional_step $x
# or in a pipeline:
@list |> map knapsack_fractional_step |> p
```

knapsack_lp_relaxation

knapsack_lp_relaxation — batch 41 builtin.

```
my $result = knapsack_lp_relaxation $x
# or in a pipeline:
@list |> map knapsack_lp_relaxation |> p
```

knapsack_unbounded_dp

knapsack_unbounded_dp — batch 41 builtin.

```
my $result = knapsack_unbounded_dp $x
# or in a pipeline:
@list |> map knapsack_unbounded_dp |> p
```

knots_to_kmh

knots_to_kmh — batch 30 builtin.

```
my $result = knots_to_kmh $input
```

kohlrausch_law

kohlrausch_law — batch 38 builtin.

```
my $result = kohlrausch_law $x
# or in a pipeline:
@list |> map kohlrausch_law |> p
```

kolmogorov_microscale

`kolmogorov_microscale` — batch 42 builtin.

```
my $result = kolmogorov_microscale $x
# or in a pipeline:
@list |> map kolmogorov_microscale |> p
```

kolmogorov_smirnov

`kolmogorov_smirnov` — batch 53 builtin.

```
my $result = kolmogorov_smirnov $x
# or in a pipeline:
@list |> map kolmogorov_smirnov |> p
```

komar_mass_step

`komar_mass_step` — batch 39 builtin.

```
my $result = komar_mass_step $x
# or in a pipeline:
@list |> map komar_mass_step |> p
```

kosaraju_scc

`kosaraju_scc` — batch 24 builtin.

```
my $result = kosaraju_scc $x
# or in a pipeline:
@list |> map kosaraju_scc |> p
```

kosaraju_step

`kosaraju_step` — batch 75 builtin.

```
my $result = kosaraju_step $x
# or in a pipeline:
@list |> map kosaraju_step |> p
```

koutecky_levich_intercept

`koutecky_levich_intercept` — batch 38 builtin.

```
my $result = koutecky_levich_intercept $x
# or in a pipeline:
@list |> map koutecky_levich_intercept |> p
```

kp

`kp` — algebraic match builtin. Alias for `keep`.

```
my $result = kp $x
# or in a pipeline:
@list |> map kp |> p
```

kp_from_kc

`kp_from_kc` — batch 21 builtin.

```
my $result = kp_from_kc $x
# or in a pipeline:
@list |> map kp_from_kc |> p
```

kpss_test

`kpss_test` — batch 78 builtin.

```
my $result = kpss_test $x
# or in a pipeline:
@list |> map kpss_test |> p
```

kpss_test_stat

`kpss_test_stat` — batch 36 builtin.

```
my $result = kpss_test_stat $x
# or in a pipeline:
@list |> map kpss_test_stat |> p
```

kraus_apply

`kraus_apply` — batch 32 builtin.

```
my $result = kraus_apply $x
# or in a pipeline:
@list |> map kraus_apply |> p
```

kretschmann_schwarzschild

`kretschmann_schwarzschild` — batch 31 builtin.

```
my $result = kretschmann_schwarzschild $x
# or in a pipeline:
@list |> map kretschmann_schwarzschild |> p
```

kron_product

`kron_product` — batch 72 builtin.

```
my $result = kron_product $x
# or in a pipeline:
@list |> map kron_product |> p
```

kronecker_four

`kronecker_four` — batch 39 builtin.

```
my $result = kronecker_four $x
# or in a pipeline:
@list |> map kronecker_four |> p
```

kronecker_three

`kronecker_three` — batch 39 builtin.

```
my $result = kronecker_three $x
# or in a pipeline:
@list |> map kronecker_three |> p
```

kruskal_h

`kruskal_h` — batch 53 builtin.

```
my $result = kruskal_h $x
# or in a pipeline:
@list |> map kruskal_h |> p
```

kruskal_step

`kruskal_step` — batch 75 builtin.

```
my $result = kruskal_step $x
# or in a pipeline:
@list |> map kruskal_step |> p
```

ks

`ks_test` (alias `ks`) — two-sample Kolmogorov-Smirnov test. Returns D statistic (max CDF difference). Like R's ks.test().

```
my $D = ks([1,2,3,4,5], [2,3,4,5,6])  # small D = similar
```

ksp_from_concs

`ksp_from_concs` — batch 21 builtin.

```
my $result = ksp_from_concs $x
# or in a pipeline:
@list |> map ksp_from_concs |> p
```

kth_largest

`kth_largest` — extended stdlib batch 3 builtin.

```
my $result = kth_largest $x
# or in a pipeline:
@list |> map kth_largest |> p
```

kth_smallest

`kth_smallest` — extended stdlib batch 3 builtin.

```
my $result = kth_smallest $x
# or in a pipeline:
@list |> map kth_smallest |> p
```

kthl

`kthl` — extended stdlib batch 3 builtin. Alias for `kth_largest`.

```
my $result = kthl $x
# or in a pipeline:
@list |> map kthl |> p
```

kths

`kths` — extended stdlib batch 3 builtin. Alias for `kth_smallest`.

```
my $result = kths $x
# or in a pipeline:
@list |> map kths |> p
```

kullback_jensen_div

`kullback_jensen_div` — batch 40 builtin.

```
my $result = kullback_jensen_div $x
# or in a pipeline:
@list |> map kullback_jensen_div |> p
```

kurt

`kurt` — math / numeric (uncategorized batch) builtin. Alias for `kurtosis`.

```
my $result = kurt $x
# or in a pipeline:
@list |> map kurt |> p
```

kurtosis

`kurtosis` — math / numeric (uncategorized batch) builtin.

```
my $result = kurtosis $x
# or in a pipeline:
@list |> map kurtosis |> p
```

kv

`kv` — batch 72 builtin.

```
my $result = kv $x
# or in a pipeline:
@list |> map kv |> p
```

kwh_to_joules

`kwh_to_joules` — batch 30 builtin.

```
my $result = kwh_to_joules $input
```

l1_coherence

`l1_coherence` — batch 32 builtin.

```
my $result = l1_coherence $x
# or in a pipeline:
@list |> map l1_coherence |> p
```

l1_norm

`l1_norm` — batch 11 builtin.

```
my $result = l1_norm $x
# or in a pipeline:
@list |> map l1_norm |> p
```

l2_norm

`l2_norm` — batch 11 builtin.

```
my $result = l2_norm $x
# or in a pipeline:
@list |> map l2_norm |> p
```

l_parallel

`l_parallel` — batch 12 builtin. Alias for `inductance_parallel`.

```
my $result = l_parallel $x
# or in a pipeline:
@list |> map l_parallel |> p
```

l_series

`l_series` — batch 12 builtin. Alias for `inductance_series`.

```
my $result = l_series $x
# or in a pipeline:
@list |> map l_series |> p
```

label_propagation

`label_propagation` — batch 75 builtin.

```
my $result = label_propagation $x
# or in a pipeline:
@list |> map label_propagation |> p
```

label_propagation_step

`label_propagation_step` — batches 14-16 builtin.

```
my $result = label_propagation_step $x
# or in a pipeline:
@list |> map label_propagation_step |> p
```

lagrange

`lagrange_interp` (aliases `lagrange`, `linterp`) performs Lagrange polynomial interpolation. Takes xs, ys, and a query point x.

```
p lagrange([0,1,2], [0,1,4], 1.5)  # 2.25
```

lagrange_l1

`lagrange_l1` — batch 31 builtin.

```
my $result = lagrange_l1 $x
# or in a pipeline:
@list |> map lagrange_l1 |> p
```

lagrangian_dual_step

`lagrangian_dual_step` — batch 41 builtin.

```
my $result = lagrangian_dual_step $x
# or in a pipeline:
@list |> map lagrangian_dual_step |> p
```

lagrangian_relax_step

`lagrangian_relax_step` — batch 41 builtin.

```
my $result = lagrangian_relax_step $x
# or in a pipeline:
@list |> map lagrangian_relax_step |> p
```

lalr_lookahead_compute

lalr_lookahead_compute — batch 68 builtin.

```
my $result = lalr_lookahead_compute $x
# or in a pipeline:
@list |> map lalr_lookahead_compute |> p
```

lambert_azimuthal_project

lambert_azimuthal_project — batches 14-16 builtin.

```
my $result = lambert_azimuthal_project $x
# or in a pipeline:
@list |> map lambert_azimuthal_project |> p
```

lambert_simple

lambert_simple — batches 14-16 builtin.

```
my $result = lambert_simple $x
# or in a pipeline:
@list |> map lambert_simple |> p
```

lambert_w

lambert_w (aliases lambertw, productlog) computes the Lambert W function (principal branch): the inverse of f(w) = w·e^w.

```
p lambertw(1)   # 0.5671 (Omega constant)
p lambertw(exp(1))  # 1.0
```

lambert_w0

lambert_w0 — batch 29 builtin.

```
my $result = lambert_w0 $x
# or in a pipeline:
@list |> map lambert_w0 |> p
```

lander_waterman_gaps

lander_waterman_gaps — batch 33 builtin.

```
my $result = lander_waterman_gaps $x
# or in a pipeline:
@list |> map lander_waterman_gaps |> p
```

laplace_de_rham_step

laplace_de_rham_step — batch 39 builtin.

```
my $result = laplace_de_rham_step $x
# or in a pipeline:
@list |> map laplace_de_rham_step |> p
```

laplace_pdf

laplace_pdf (alias laplacepdf) evaluates the Laplace distribution PDF at x with location mu and scale b.

```
p laplacepdf(0, 0, 1)  # 0.5 (peak)
```

laplacian_kernel_value

`laplacian_kernel_value` — batch 40 builtin.

```
my $result = laplacian_kernel_value $x
# or in a pipeline:
@list |> map laplacian_kernel_value |> p
```

larmor_frequency

`larmor_frequency` — batch 32 builtin.

```
my $result = larmor_frequency $x
# or in a pipeline:
@list |> map larmor_frequency |> p
```

last_arg

`last_arg` — functional primitives builtin.

```
my $result = last_arg $x
# or in a pipeline:
@list |> map last_arg |> p
```

last_clj

`last_clj` — algebraic match builtin.

```
my $result = last_clj $x
# or in a pipeline:
@list |> map last_clj |> p
```

last_elem

`last_elem` — list helpers (batch 4) builtin.

```
my $result = last_elem $x
# or in a pipeline:
@list |> map last_elem |> p
```

last_eq

`last_eq` — collection (batch 2) builtin.

```
my $result = last_eq $x
# or in a pipeline:
@list |> map last_eq |> p
```

last_index_of

`last_index_of` — collection (batch 2) builtin.

```
my $result = last_index_of $x
# or in a pipeline:
@list |> map last_index_of |> p
```

last_of_month

`last_of_month` — batch 9 builtin.

```
my $result = last_of_month $x
# or in a pipeline:
@list |> map last_of_month |> p
```

last_word

last_word — string (batch 2) builtin.

```
my $result = last_word $x
# or in a pipeline:
@list |> map last_word |> p
```

lastc

lastc — algebraic match builtin. Alias for last_clj.

```
my $result = lastc $x
# or in a pipeline:
@list |> map lastc |> p
```

lat_lng_to_tile_xyz

lat_lng_to_tile_xyz — batch 58 builtin.

```
my $result = lat_lng_to_tile_xyz $input
```

lat_lng_to_xy_lambert

lat_lng_to_xy_lambert — batch 58 builtin.

```
my $result = lat_lng_to_xy_lambert $input
```

lat_lng_to_xy_mercator

lat_lng_to_xy_mercator — batch 58 builtin.

```
my $result = lat_lng_to_xy_mercator $input
```

lax_friedrichs_flux

lax_friedrichs_flux — batch 34 builtin.

```
my $result = lax_friedrichs_flux $x
# or in a pipeline:
@list |> map lax_friedrichs_flux |> p
```

lax_wendroff_flux

lax_wendroff_flux — batch 34 builtin.

```
my $result = lax_wendroff_flux $x
# or in a pipeline:
@list |> map lax_wendroff_flux |> p
```

lazy_caterer

lazy_caterer — batch 11 builtin.

```
my $result = lazy_caterer $x
# or in a pipeline:
@list |> map lazy_caterer |> p
```

lb_to_kg

`lb_to_kg` — batch 30 builtin.

```
my $result = lb_to_kg $input
```

lbf_to_newtons

`lbf_to_newtons` — file stat / path builtin.

```
my $result = lbf_to_newtons $input
```

lbound

`lbound` — extended stdlib builtin. Alias for `lower_bound`.

```
my $result = lbound $x
# or in a pipeline:
@list |> map lbound |> p
```

lbs_to_kg

`lbs_to_kg` — unit conversions builtin.

```
my $result = lbs_to_kg $input
```

lc_frequency

`lc_frequency` — batch 23 builtin.

```
my $result = lc_frequency $x
# or in a pipeline:
@list |> map lc_frequency |> p
```

lc_lines

`lc_lines` — trivial string ops builtin. Alias for `line_count`.

```
my $result = lc_lines $x
# or in a pipeline:
@list |> map lc_lines |> p
```

lc_omega

`lc_omega` — batch 23 builtin.

```
my $result = lc_omega $x
# or in a pipeline:
@list |> map lc_omega |> p
```

lc_resonant

`lc_resonant` — batch 12 builtin.

```
my $result = lc_resonant $x
# or in a pipeline:
@list |> map lc_resonant |> p
```

lcdm_eos

`lcdm_eos` — batch 31 builtin.

```
my $result = lcdm_eos $x
# or in a pipeline:
@list |> map lcdm_eos |> p
```

lcg_next_u32

`lcg_next_u32` — batch 9 builtin.

```
my $result = lcg_next_u32 $x
# or in a pipeline:
@list |> map lcg_next_u32 |> p
```

lcg_numrec_step

`lcg_numrec_step` — batch 26 builtin.

```
my $result = lcg_numrec_step $x
# or in a pipeline:
@list |> map lcg_numrec_step |> p
```

lcm

`lcm` — trivial numeric / predicate builtins builtin.

```
my $result = lcm $x
# or in a pipeline:
@list |> map lcm |> p
```

lcontract

`length_contraction($proper_length, $velocity)` (alias `lcontract`) — contracted length $L = L_0 \times \sqrt{1-v^2/c^2}$.

```
p lcontract(1, 0.99 * 3e8)   # ~0.14 m (1m at 99% c)
```

lcount

`lcount` — extended stdlib batch 3 builtin. Alias for `count_lines`.

```
my $result = lcount $x
# or in a pipeline:
@list |> map lcount |> p
```

lcp

`lcp` — string (batch 2) builtin. Alias for `longest_common_prefix`.

```
my $result = lcp $x
# or in a pipeline:
@list |> map lcp |> p
```

lcp_array

`lcp_array` — batch 10 builtin.

```
my $result = lcp_array $x
# or in a pipeline:
@list |> map lcp_array |> p
```

lcs_length

`lcs_length` — batch 33 builtin.

```
my $result = lcs_length $x
# or in a pipeline:
@list |> map lcs_length |> p
```

lcseq

`lcseq` — extended stdlib batch 3 builtin. Alias for `longest_common_subsequence`.

```
my $result = lcseq $x
# or in a pipeline:
@list |> map lcseq |> p
```

lcsub

`lcsub` — extended stdlib batch 3 builtin. Alias for `longest_common_substring`.

```
my $result = lcsub $x
# or in a pipeline:
@list |> map lcsub |> p
```

ld_d

`ld_d` — batch 33 builtin.

```
my $result = ld_d $x
# or in a pipeline:
@list |> map ld_d |> p
```

ld_r_squared

`ld_r_squared` — batch 33 builtin.

```
my $result = ld_r_squared $x
# or in a pipeline:
@list |> map ld_r_squared |> p
```

ldec

`ldec` — extended stdlib batch 3 builtin. Alias for `longest_decreasing`.

```
my $result = ldec $x
# or in a pipeline:
@list |> map ldec |> p
```

le_chatelier_dir

`le_chatelier_dir` — batch 21 builtin.

```
my $result = le_chatelier_dir $x
# or in a pipeline:
@list |> map le_chatelier_dir |> p
```

leading_zeros

`leading_zeros` — base conversion builtin.

```
my $result = leading_zeros $x
# or in a pipeline:
@list |> map leading_zeros |> p
```

leaky_relu

`leaky_relu` (alias `lrelu`) applies Leaky ReLU: x if x⩾0, alpha*x otherwise (default alpha=0.01).

```
p lrelu(5)     # 5
p lrelu(-5)    # -0.05
p lrelu(-5, 0.1)  # -0.5
```

leap_year_gregorian

`leap_year_gregorian` — batch 52 builtin.

```
my $result = leap_year_gregorian $x
# or in a pipeline:
@list |> map leap_year_gregorian |> p
```

leapfrog_step

`leapfrog_step` — batch 34 builtin.

```
my $result = leapfrog_step $x
# or in a pipeline:
@list |> map leapfrog_step |> p
```

least_common

`least_common` — list helpers (batch 4) builtin.

```
my $result = least_common $x
# or in a pipeline:
@list |> map least_common |> p
```

leet

`leet` — string processing (uncategorized batch) builtin. Alias for `leetspeak`.

```
my $result = leet $x
# or in a pipeline:
@list |> map leet |> p
```

leetspeak

`leetspeak` — string processing (uncategorized batch) builtin.

```
my $result = leetspeak $x
# or in a pipeline:
@list |> map leetspeak |> p
```

left_str

`left_str` — string (batch 2) builtin.

```
my $result = left_str $x
# or in a pipeline:
@list |> map left_str |> p
```

lehmer_mean

`lehmer_mean` — batch 11 builtin.

```
my $result = lehmer_mean $x
# or in a pipeline:
@list |> map lehmer_mean |> p
```

lemmatize_lemmy

`lemmatize_lemmy` — batch 69 builtin.

```
my $result = lemmatize_lemmy $x
# or in a pipeline:
@list |> map lemmatize_lemmy |> p
```

lemmatize_wordnet

`lemmatize_wordnet` — batch 69 builtin.

```
my $result = lemmatize_wordnet $x
# or in a pipeline:
@list |> map lemmatize_wordnet |> p
```

length_each

`length_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = length_each $x
# or in a pipeline:
@list |> map length_each |> p
```

lens_focal_length

`lens_focal_length` — batch 12 builtin.

```
my $result = lens_focal_length $x
# or in a pipeline:
@list |> map lens_focal_length |> p
```

lens_magnification

`lens_magnification` — batch 23 builtin.

```
my $result = lens_magnification $x
# or in a pipeline:
@list |> map lens_magnification |> p
```

lens_power

`lens_power` — physics formulas builtin.

```
my $result = lens_power $x
# or in a pipeline:
@list |> map lens_power |> p
```

lense_thirring_omega

`lense_thirring_omega` — batch 31 builtin.

```
my $result = lense_thirring_omega $x
# or in a pipeline:
@list |> map lense_thirring_omega |> p
```

lensing_convergence

`lensing_convergence` — batch 31 builtin.

```
my $result = lensing_convergence $x
# or in a pipeline:
@list |> map lensing_convergence |> p
```

lensmaker

`lensmaker` — batch 23 builtin.

```
my $result = lensmaker $x
# or in a pipeline:
@list |> map lensmaker |> p
```

leontief_input

`leontief_input` — batch 73 builtin.

```
my $result = leontief_input $x
# or in a pipeline:
@list |> map leontief_input |> p
```

leontief_output

`leontief_output` — batch 73 builtin.

```
my $result = leontief_output $x
# or in a pipeline:
@list |> map leontief_output |> p
```

lerner_index

`lerner_index` — batch 73 builtin.

```
my $result = lerner_index $x
# or in a pipeline:
@list |> map lerner_index |> p
```

lerp

`lerp($a, $b, $t)` — linear interpolation between a and b. When t=0 returns a, t=1 returns b, t=0.5 returns midpoint.

```
p lerp(0, 100, 0.5)   # 50
p lerp(10, 20, 0.25)  # 12.5
# Animation: smoothly transition between values
for my $t (0:10) { p lerp($start, $end, $t/10) }
```

lerp_color

`lerp_color` — batch 9 builtin. Alias for `color_blend_t`.

```
my $result = lerp_color $x
# or in a pipeline:
@list |> map lerp_color |> p
```

leslie_step

`leslie_step` — batch 22 builtin.

```
my $result = leslie_step $x
# or in a pipeline:
@list |> map leslie_step |> p
```

less

`LIST |> pager` / `pager LIST` — pipe each element (one per line) into the user's `$PAGER` (default `less -R`; falls back to `more`, then plain stdout). Bypasses the pager when stdout isn't a TTY so pipelines like `stryke '... |> pager' | grep` still compose.

Blocks until the user quits the pager; returns `undef`.

Aliases: `pager`, `pg`, `less`.

```
# browse every callable spelling interactively
keys %all |> sort |> pager

# filter the reference for parallel ops
keys %b |> grep { $b{_} eq "parallel" } |> pager

# whole file, one screen at a time
slurp("README.md") |> pager
```

letter_frequency

`letter_frequency` — batch 26 builtin.

```
my $result = letter_frequency $x
# or in a pipeline:
@list |> map letter_frequency |> p
```

letters_lc

`letters_lc` extracts only lowercase letters from a string.

```
p join "", letters_lc("Hello World 123")  # elloorld
```

letters_uc

`letters_uc` extracts only uppercase letters from a string.

```
p join "", letters_uc("Hello World 123")  # HW
```

lev

`lev` — extended stdlib builtin. Alias for `levenshtein`.

```
my $result = lev $x
# or in a pipeline:
@list |> map lev |> p
```

level_k_step

`level_k_step` — batch 43 builtin.

```
my $result = level_k_step $x
# or in a pipeline:
@list |> map level_k_step |> p
```

level_premium

`level_premium` — batch 60 builtin.

```
my $result = level_premium $x
# or in a pipeline:
@list |> map level_premium |> p
```

levenshtein

`levenshtein` — extended stdlib builtin.

```
my $result = levenshtein $x
# or in a pipeline:
@list |> map levenshtein |> p
```

levenshtein_dist

`levenshtein_dist` — batch 70 builtin.

```
my $result = levenshtein_dist $x
# or in a pipeline:
@list |> map levenshtein_dist |> p
```

leverage_index

`leverage_index` — batch 56 builtin.

```
my $result = leverage_index $x
# or in a pipeline:
@list |> map leverage_index |> p
```

levi_civita_four

`levi_civita_four` — batch 39 builtin.

```
my $result = levi_civita_four $x
# or in a pipeline:
@list |> map levi_civita_four |> p
```

levi_civita_three

`levi_civita_three` — batch 39 builtin.

```
my $result = levi_civita_three $x
# or in a pipeline:
@list |> map levi_civita_three |> p
```

levich_current_rde

`levich_current_rde` — batch 38 builtin.

```
my $result = levich_current_rde $x
# or in a pipeline:
@list |> map levich_current_rde |> p
```

levin_t_transform

`levin_t_transform` — batches 14-16 builtin.

```
my $result = levin_t_transform $x
# or in a pipeline:
@list |> map levin_t_transform |> p
```

levy_pdf

`levy_pdf` — batches 17-18 builtin.

```
my $result = levy_pdf $x
# or in a pipeline:
@list |> map levy_pdf |> p
```

lewis_dot_count

`lewis_dot_count` — batch 63 builtin.

```
my $result = lewis_dot_count $x
# or in a pipeline:
@list |> map lewis_dot_count |> p
```

lex_keyword_classify

`lex_keyword_classify` — batch 68 builtin.

```
my $result = lex_keyword_classify $x
# or in a pipeline:
@list |> map lex_keyword_classify |> p
```

lexicalized_parse

`lexicalized_parse` — batch 69 builtin.

```
my $result = lexicalized_parse $x
# or in a pipeline:
@list |> map lexicalized_parse |> p
```

lfilter_zi_init

`lfilter_zi_init` — batch 74 builtin.

```
my $result = lfilter_zi_init $x
# or in a pipeline:
@list |> map lfilter_zi_init |> p
```

lfrm

`lfrm` — go/general functional utilities builtin. Alias for `lines_from`.

```
my $result = lfrm $x
# or in a pipeline:
@list |> map lfrm |> p
```

lfsr_galois_step

`lfsr_galois_step` — batch 26 builtin.

```
my $result = lfsr_galois_step $x
# or in a pipeline:
@list |> map lfsr_galois_step |> p
```

lie_derivative_scalar_step

`lie_derivative_scalar_step` — batch 39 builtin.

```
my $result = lie_derivative_scalar_step $x
# or in a pipeline:
@list |> map lie_derivative_scalar_step |> p
```

lie_derivative_vector_step

`lie_derivative_vector_step` — batch 39 builtin.

```
my $result = lie_derivative_vector_step $x
# or in a pipeline:
@list |> map lie_derivative_vector_step |> p
```

lie_split

`lie_split` — batch 34 builtin.

```
my $result = lie_split $x
# or in a pipeline:
@list |> map lie_split |> p
```

life_expectancy_e0

`life_expectancy_e0` — batch 60 builtin.

```
my $result = life_expectancy_e0 $x
# or in a pipeline:
@list |> map life_expectancy_e0 |> p
```

light_deflection_angle

`light_deflection_angle` — batch 31 builtin.

```
my $result = light_deflection_angle $x
# or in a pipeline:
@list |> map light_deflection_angle |> p
```

light_year

`light_year` (alias `ly`) — ly ≈ 9.461×10^{15} m. Distance light travels in one year.

```
p ly             # 9.4607304725808e15 m
p ly / au        # ~63241 AU
```

likelihood_ratio_test

`likelihood_ratio_test` — batch 36 builtin.

```
my $result = likelihood_ratio_test $x
# or in a pipeline:
@list |> map likelihood_ratio_test |> p
```

limiting_current_density

`limiting_current_density` — batch 38 builtin.

```
my $result = limiting_current_density $x
# or in a pipeline:
@list |> map limiting_current_density |> p
```

lin_bairstow_step

`lin_bairstow_step` — batch 10 builtin.

```
my $result = lin_bairstow_step $x
# or in a pipeline:
@list |> map lin_bairstow_step |> p
```

lin_kernighan_step

`lin_kernighan_step` — batch 41 builtin.

```
my $result = lin_kernighan_step $x
# or in a pipeline:
@list |> map lin_kernighan_step |> p
```

linc

`linc` — extended stdlib batch 3 builtin. Alias for `longest_increasing`.

```
my $result = linc $x
# or in a pipeline:
@list |> map linc |> p
```

line_count

`line_count` — trivial string ops builtin.

```
my $result = line_count $x
# or in a pipeline:
@list |> map line_count |> p
```

line_count_simple

`line_count_simple` — batch 11 builtin. Alias for `string_lines_count`.

```
my $result = line_count_simple $x
# or in a pipeline:
@list |> map line_count_simple |> p
```

line_intersection

`line_intersection($x1, $y1, $x2, $y2, $x3, $y3, $x4, $y4)` — finds the intersection point of two lines defined by points (x1,y1)-(x2,y2) and (x3,y3)-(x4,y4). Returns [$x, $y] or undef if parallel.

```
my $pt = line_intersection(0, 0, 1, 1, 0, 1, 1, 0)
p @$pt   # (0.5, 0.5)
```

linear_entropy

`linear_entropy` — batch 32 builtin.

```
my $result = linear_entropy $x
# or in a pipeline:
@list |> map linear_entropy |> p
```

linear_regression

`linear_regression` — math / numeric (uncategorized batch) builtin.

```
my $result = linear_regression $x
# or in a pipeline:
@list |> map linear_regression |> p
```

linear_search

`linear_search` — collection (batch 2) builtin.

```
my $result = linear_search $x
# or in a pipeline:
@list |> map linear_search |> p
```

linearized_pb_step

`linearized_pb_step` — batch 38 builtin.

```
my $result = linearized_pb_step $x
# or in a pipeline:
@list |> map linearized_pb_step |> p
```

lineintr

`lineintr` — geometry (extended) builtin. Alias for `line_intersection`.

```
my $result = lineintr $x
# or in a pipeline:
@list |> map lineintr |> p
```

lines_from

`lines_from` — go/general functional utilities builtin.

```
my $result = lines_from $x
# or in a pipeline:
@list |> map lines_from |> p
```

lineweaver_burk

`lineweaver_burk` — batch 12 builtin.

```
my $result = lineweaver_burk $x
# or in a pipeline:
@list |> map lineweaver_burk |> p
```

linf_norm

`linf_norm` — batch 11 builtin.

```
my $result = linf_norm $x
# or in a pipeline:
@list |> map linf_norm |> p
```

linking_number_two

`linking_number_two` — batch 37 builtin.

```
my $result = linking_number_two $x
# or in a pipeline:
@list |> map linking_number_two |> p
```

linreg

`linreg` — math / numeric (uncategorized batch) builtin. Alias for `linear_regression`.

```
my $result = linreg $x
# or in a pipeline:
@list |> map linreg |> p
```

linspace

`linspace` — matrix / linear algebra builtin.

```
my $result = linspace $x
# or in a pipeline:
@list |> map linspace |> p
```

liouville

`liouville` — batch 35 builtin.

```
my $result = liouville $x
# or in a pipeline:
@list |> map liouville |> p
```

liquefaction_potential_index

`liquefaction_potential_index` — batch 65 builtin.

```
my $result = liquefaction_potential_index $x
# or in a pipeline:
@list |> map liquefaction_potential_index |> p
```

list_coloring_step

`list_coloring_step` — batch 41 builtin.

```
my $result = list_coloring_step $x
# or in a pipeline:
@list |> map list_coloring_step |> p
```

list_compact

`list_compact` — list helpers (batch 4) builtin.

```
my $result = list_compact $x
# or in a pipeline:
@list |> map list_compact |> p
```

list_count

`list_count` (alias `list_size`) returns the total number of elements after flattening a nested list structure. Unlike `count` which returns the top-level element count, `list_count` recursively descends into array references and counts only leaf values. This is useful when you have a list of lists and need to know the total number of individual items rather than the number of sublists.

```
my @nested = ([1, 2], [3, 4, 5], [6])
p list_count @nested   # 6
my @deep = ([1, [2, 3]], [4])
p list_size @deep      # 4
p list_count 1, 2, 3   # 3  (flat list works too)
```

list_difference

`list_difference` — collection (batch 2) builtin. Alias for `array_difference`.

```
my $result = list_difference $x
# or in a pipeline:
@list |> map list_difference |> p
```

list_eq

`list_eq` — list helpers (batch 4) builtin.

```
my $result = list_eq $x
# or in a pipeline:
@list |> map list_eq |> p
```

list_flatten_deep

`list_flatten_deep` — list helpers (batch 4) builtin.

```
my $result = list_flatten_deep $x
# or in a pipeline:
@list |> map list_flatten_deep |> p
```

list_intersection

`list_intersection` — collection (batch 2) builtin. Alias for `array_intersection`.

```
my $result = list_intersection $x
# or in a pipeline:
@list |> map list_intersection |> p
```

list_reverse

`list_reverse` — file stat / path builtin. Alias for `reverse_list`.

```
my $result = list_reverse $x
# or in a pipeline:
@list |> map list_reverse |> p
```

list_scheduling_step

`list_scheduling_step` — batch 41 builtin.

```
my $result = list_scheduling_step $x
# or in a pipeline:
@list |> map list_scheduling_step |> p
```

list_union

`list_union` — collection (batch 2) builtin. Alias for `array_union`.

```
my $result = list_union $x
# or in a pipeline:
@list |> map list_union |> p
```

liters_to_gallons

`liters_to_gallons` — file stat / path builtin.

```
my $result = liters_to_gallons $input
```

liters_to_ml

`liters_to_ml` — file stat / path builtin.

```
my $result = liters_to_ml $input
```

lithium_ion_diffusivity

`lithium_ion_diffusivity` — batch 38 builtin.

```
my $result = lithium_ion_diffusivity $x
# or in a pipeline:
@list |> map lithium_ion_diffusivity |> p
```

lix

`lix` — batch 9 builtin.

```
my $result = lix $x
# or in a pipeline:
@list |> map lix |> p
```

ljt

`ljt` — extended stdlib builtin. Alias for `ljust_text`.

```
my $result = ljt $x
# or in a pipeline:
@list |> map ljt |> p
```

ljung_box_q

`ljung_box_q` — batch 78 builtin.

```
my $result = ljung_box_q $x
# or in a pipeline:
@list |> map ljung_box_q |> p
```

ljust

`ljust` — string helpers (batch 4) builtin.

```
my $result = ljust $x
# or in a pipeline:
@list |> map ljust |> p
```

ljust_text

`ljust_text` — extended stdlib builtin.

```
my $result = ljust_text $x
# or in a pipeline:
@list |> map ljust_text |> p
```

lkpa

`lkpa` — haskell list functions builtin. Alias for `lookup_assoc`.

```
my $result = lkpa $x
# or in a pipeline:
@list |> map lkpa |> p
```

ll1_first_set

`ll1_first_set` — batch 68 builtin.

```
my $result = ll1_first_set $x
# or in a pipeline:
@list |> map ll1_first_set |> p
```

ll1_follow_set

`ll1_follow_set` — batch 68 builtin.

```
my $result = ll1_follow_set $x
# or in a pipeline:
@list |> map ll1_follow_set |> p
```

ll1_predict_table

`ll1_predict_table` — batch 68 builtin.

```
my $result = ll1_predict_table $x
# or in a pipeline:
@list |> map ll1_predict_table |> p
```

lll_2x2_step

`lll_2x2_step` — batch 35 builtin.

```
my $result = lll_2x2_step $x
# or in a pipeline:
@list |> map lll_2x2_step |> p
```

lm

`lm_fit` (alias `lm`) — simple linear regression. Returns hash with intercept, slope, r_squared, residuals, fitted. Like R's lm().

```
my %m = %{lm([1,2,3,4,5], [2,4,5,4,5])}
p $m{slope}       # slope
p $m{r_squared}   # R²
```

lm_step

`lm_step` — batch 19 builtin.

```
my $result = lm_step $x
# or in a pipeline:
@list |> map lm_step |> p
```

ln10

`ln10` — math / physics constants builtin.

```
my $result = ln10 $x
# or in a pipeline:
@list |> map ln10 |> p
```

ln2

`ln2` — math / physics constants builtin.

```
my $result = ln2 $x
# or in a pipeline:
@list |> map ln2 |> p
```

lnormpdf

`lognormal_pdf` (alias `lnormpdf`) evaluates the log-normal distribution PDF at x with parameters mu and sigma.

```
p lnormpdf(1.0, 0, 1)  # LogN(0,1) at x=1
```

lns

`lns` — math / numeric (uncategorized batch) builtin. Alias for `look_and_say`.

```
my $result = lns $x
# or in a pipeline:
@list |> map lns |> p
```

load_avg

`load_avg` — system load averages as a 3-element arrayref `[1min, 5min, 15min]`. Uses `getloadavg()` on unix. Returns `undef` on unsupported platforms.

```
my $la = load_avg
p "1m=$la->[0] 5m=$la->[1] 15m=$la->[2]"
load_avg |> spark |> p                  # sparkline of load
```

load_centrality

`load_centrality` — batch 75 builtin.

```
my $result = load_centrality $x
# or in a pipeline:
@list |> map load_centrality |> p
```

loan_balance

`loan_balance` — batch 13 builtin.

```
my $result = loan_balance $x
# or in a pipeline:
@list |> map loan_balance |> p
```

loan_payment_pmt

`loan_payment_pmt` — batch 13 builtin.

```
my $result = loan_payment_pmt $x
# or in a pipeline:
@list |> map loan_payment_pmt |> p
```

lobatto_iiia_2

`lobatto_iiia_2` — batch 34 builtin.

```
my $result = lobatto_iiia_2 $input
```

lobatto_iiic_3

`lobatto_iiic_3` — batch 34 builtin.

```
my $result = lobatto_iiic_3 $input
```

local_efficiency

`local_efficiency` — batches 14-16 builtin.

```
my $result = local_efficiency $x
# or in a pipeline:
@list |> map local_efficiency |> p
```

local_search_swap_step

`local_search_swap_step` — batch 41 builtin.

```
my $result = local_search_swap_step $x
# or in a pipeline:
@list |> map local_search_swap_step |> p
```

loess

`lowess` (alias `loess`) — locally-weighted scatterplot smoothing (LOWESS/LOESS). Returns smoothed Y values with tricube weighting.

```
my @smooth = @{lowess([1,2,3,4,5], [2,1,4,3,5])}
```

log10

`log10` — trivial numeric / predicate builtins builtin.

```
my $result = log10 $x
# or in a pipeline:
@list |> map log10 |> p
```

log1p_exp

`log1p_exp` — batch 11 builtin.

```
my $result = log1p_exp $x
# or in a pipeline:
@list |> map log1p_exp |> p
```

log2

`log2` — trivial numeric / predicate builtins builtin.

```
my $result = log2 $x
# or in a pipeline:
@list |> map log2 |> p
```

log_base

`log_base` — math functions builtin.

```
my $result = log_base $x
# or in a pipeline:
@list |> map log_base |> p
```

log_gamma_simple

`log_gamma_simple` — batches 17-18 builtin.

```
my $result = log_gamma_simple $x
# or in a pipeline:
@list |> map log_gamma_simple |> p
```

log_law_wind_profile

`log_law_wind_profile` — batch 42 builtin.

```
my $result = log_law_wind_profile $x
# or in a pipeline:
@list |> map log_law_wind_profile |> p
```

log_loss

`log_loss` — batch 27 builtin.

```
my $result = log_loss $x
# or in a pipeline:
@list |> map log_loss |> p
```

log_sigmoid

`log_sigmoid` — batch 11 builtin.

```
my $result = log_sigmoid $x
# or in a pipeline:
@list |> map log_sigmoid |> p
```

log_softmax

`log_softmax` — batch 27 builtin.

```
my $result = log_softmax $x
# or in a pipeline:
@list |> map log_softmax |> p
```

log_sum_exp

`log_sum_exp` — batch 11 builtin.

```
my $result = log_sum_exp $x
# or in a pipeline:
@list |> map log_sum_exp |> p
```

log_with_base

`log_with_base` — extended stdlib builtin.

```
my $result = log_with_base $x
# or in a pipeline:
@list |> map log_with_base |> p
```

logb

`logb` — extended stdlib builtin. Alias for `log_with_base`.

```
my $result = logb $x
# or in a pipeline:
@list |> map logb |> p
```

loggamma

`loggamma` — batch 72 builtin.

```
my $result = loggamma $x
# or in a pipeline:
@list |> map loggamma |> p
```

logistic_growth_analytic

`logistic_growth_analytic` — batch 22 builtin.

```
my $result = logistic_growth_analytic $x
# or in a pipeline:
@list |> map logistic_growth_analytic |> p
```

logistic_growth_step

`logistic_growth_step` — batch 22 builtin.

```
my $result = logistic_growth_step $x
# or in a pipeline:
@list |> map logistic_growth_step |> p
```

logistic_loss

`logistic_loss` — batch 27 builtin.

```
my $result = logistic_loss $x
# or in a pipeline:
@list |> map logistic_loss |> p
```

logit_log_likelihood

`logit_log_likelihood` — batch 36 builtin.

```
my $result = logit_log_likelihood $x
# or in a pipeline:
@list |> map logit_log_likelihood |> p
```

lomax_pdf

`lomax_pdf` — batches 17-18 builtin.

```
my $result = lomax_pdf $x
# or in a pipeline:
@list |> map lomax_pdf |> p
```

lombscargle_freq

`lombscargle_freq` — batch 74 builtin.

```
my $result = lombscargle_freq $x
# or in a pipeline:
@list |> map lombscargle_freq |> p
```

longest

`longest` — collection (batch 2) builtin.

```
my $result = longest $x
# or in a pipeline:
@list |> map longest |> p
```

longest_common_prefix

`longest_common_prefix` — string (batch 2) builtin.

```
my $result = longest_common_prefix $x
# or in a pipeline:
@list |> map longest_common_prefix |> p
```

longest_common_subsequence

`longest_common_subsequence` — extended stdlib batch 3 builtin.

```
my $result = longest_common_subsequence $x
# or in a pipeline:
@list |> map longest_common_subsequence |> p
```

longest_common_substring

`longest_common_substring` — extended stdlib batch 3 builtin.

```
my $result = longest_common_substring $x
# or in a pipeline:
@list |> map longest_common_substring |> p
```

longest_decreasing

`longest_decreasing` — extended stdlib batch 3 builtin.

```
my $result = longest_decreasing $x
# or in a pipeline:
@list |> map longest_decreasing |> p
```

longest_increasing

`longest_increasing` — extended stdlib batch 3 builtin.

```
my $result = longest_increasing $x
# or in a pipeline:
@list |> map longest_increasing |> p
```

longest_run

`longest_run` — extended stdlib batch 3 builtin.

```
my $result = longest_run $x
# or in a pipeline:
@list |> map longest_run |> p
```

look_and_say

`look_and_say` — math / numeric (uncategorized batch) builtin.

```
my $result = look_and_say $x
# or in a pipeline:
@list |> map look_and_say |> p
```

lookback_call

`lookback_call` — batch 20 builtin.

```
my $result = lookback_call $x
# or in a pipeline:
@list |> map lookback_call |> p
```

lookback_time

`lookback_time` — batch 31 builtin.

```
my $result = lookback_time $x
# or in a pipeline:
@list |> map lookback_time |> p
```

lookup_assoc

`lookup_assoc` — haskell list functions builtin.

```
my $result = lookup_assoc $x
# or in a pipeline:
@list |> map lookup_assoc |> p
```

lorentz

`lorentz_factor($velocity)` (alias `lorentz`) — relativistic gamma factor $\gamma = 1/\sqrt{(1-v^2/c^2)}$. Returns infinity at v ≥ c.

```
p lorentz(0)          # 1 (at rest)
p lorentz(0.9 * 3e8)  # ~2.29 (at 90% speed of light)
```

lorentz_factor_v

`lorentz_factor_v` — batch 12 builtin.

```
my $result = lorentz_factor_v $x
# or in a pipeline:
@list |> map lorentz_factor_v |> p
```

lorentz_force_mag

`lorentz_force_mag` — batch 23 builtin.

```
my $result = lorentz_force_mag $x
# or in a pipeline:
@list |> map lorentz_force_mag |> p
```

lorentz_gamma

`lorentz_gamma` — batch 23 builtin.

```
my $result = lorentz_gamma $x
# or in a pipeline:
@list |> map lorentz_gamma |> p
```

lorenz

`lorenz_curve(@data)` (alias `lorenz`) — computes the Lorenz curve as an arrayref of [cumulative_share_of_population, cumulative_share_of_wealth] pairs. Used to visualize inequality. The diagonal represents perfect equality.

```
my $curve = lorenz(10, 20, 30, 40)
for my $pt (@$curve) {
    p "$pt->[0], $pt->[1]"
}
```

lose_lose

`lose_lose` — batch 81 builtin.

```
my $result = lose_lose $x
# or in a pipeline:
@list |> map lose_lose |> p
```

loss_frequency_poisson

`loss_frequency_poisson` — batch 60 builtin.

```
my $result = loss_frequency_poisson $x
# or in a pipeline:
@list |> map loss_frequency_poisson |> p
```

loss_severity_lognormal

`loss_severity_lognormal` — batch 60 builtin.

```
my $result = loss_severity_lognormal $x
# or in a pipeline:
@list |> map loss_severity_lognormal |> p
```

lotka_volterra_step

`lotka_volterra_step` — batch 12 builtin.

```
my $result = lotka_volterra_step $x
# or in a pipeline:
@list |> map lotka_volterra_step |> p
```

louvain_gain

`louvain_gain` — batch 75 builtin.

```
my $result = louvain_gain $x
# or in a pipeline:
@list |> map louvain_gain |> p
```

lower_bound

`lower_bound` — extended stdlib builtin.

```
my $result = lower_bound $x
# or in a pipeline:
@list |> map lower_bound |> p
```

lower_triangular_q

`lower_triangular_q` — batches 17-18 builtin.

```
my $result = lower_triangular_q $x
# or in a pipeline:
@list |> map lower_triangular_q |> p
```

lowercase

`lowercase` — string (batch 2) builtin.

```
my $result = lowercase $x
# or in a pipeline:
@list |> map lowercase |> p
```

lowpass_filter

`lowpass_filter(\@signal, $cutoff)` (alias `lpf`) — applies a simple low-pass filter to remove high-frequency components. Cutoff is normalized (0-1). Uses exponential moving average.

```
my @noisy = map { sin(_/10) + rand(0.2) } 1:100
my $smooth = lpf(\@noisy, 0.1)
```

lp_norm

`lp_norm` — batch 11 builtin.

```
my $result = lp_norm $x
# or in a pipeline:
@list |> map lp_norm |> p
```

lp_relax_round

`lp_relax_round` — batch 41 builtin.

```
my $result = lp_relax_round $x
# or in a pipeline:
@list |> map lp_relax_round |> p
```

lpad

`lpad` — trivial string ops builtin. Alias for `pad_left`.

```
my $result = lpad $x
# or in a pipeline:
@list |> map lpad |> p
```

lplanck

`planck_length` (alias `lplanck`) — $l_P \approx 1.616 \times 10^{-35}$ m. Smallest meaningful length in quantum mechanics.

```
p lplanck   # 1.616255e-35 m
```

lpt

`lpt` — batch 10 builtin. Alias for `lpt_schedule`.

```
my $result = lpt $x
# or in a pipeline:
@list |> map lpt |> p
```

lpt_schedule

`lpt_schedule` — batch 10 builtin.

```
my $result = lpt_schedule $x
# or in a pipeline:
@list |> map lpt_schedule |> p
```

lpush

`lpush` — batch 71 builtin.

```
my $result = lpush $x
# or in a pipeline:
@list |> map lpush |> p
```

lqg_step

`lqg_step` — batch 59 builtin.

```
my $result = lqg_step $x
# or in a pipeline:
@list |> map lqg_step |> p
```

lqr_gain_continuous

`lqr_gain_continuous` — batch 59 builtin.

```
my $result = lqr_gain_continuous $x
# or in a pipeline:
@list |> map lqr_gain_continuous |> p
```

lqr_gain_discrete

`lqr_gain_discrete` — batch 59 builtin.

```
my $result = lqr_gain_discrete $x
# or in a pipeline:
@list |> map lqr_gain_discrete |> p
```

lr0_items_step

`lr0_items_step` — batch 68 builtin.

```
my $result = lr0_items_step $x
# or in a pipeline:
@list |> map lr0_items_step |> p
```

lr1_canonical_collection

`lr1_canonical_collection` — batch 68 builtin.

```
my $result = lr1_canonical_collection $x
# or in a pipeline:
@list |> map lr1_canonical_collection |> p
```

lrange

`lrange` — batch 71 builtin.

```
my $result = lrange $x
# or in a pipeline:
@list |> map lrange |> p
```

lrem

`lrem` — batch 71 builtin.

```
my $result = lrem $x
# or in a pipeline:
@list |> map lrem |> p
```

lru_new

`lru_new` — data structure helpers builtin.

```
my $result = lru_new $x
# or in a pipeline:
@list |> map lru_new |> p
```

lrun

`lrun` — extended stdlib batch 3 builtin. Alias for `longest_run`.

```
my $result = lrun $x
# or in a pipeline:
@list |> map lrun |> p
```

lse

`lse` — batch 11 builtin. Alias for `log_sum_exp`.

```
my $result = lse $x
# or in a pipeline:
@list |> map lse |> p
```

lsearch

`lsearch` — collection (batch 2) builtin. Alias for `linear_search`.

```
my $result = lsearch $x
# or in a pipeline:
@list |> map lsearch |> p
```

lsp_completion_words

`lsp_completion_words` (alias `lsp_words`) — emits every name LSP tab-complete should know about, sorted ASCII. Three sources merged: (1) every callable bare-name spelling from `%all` (~9k names, primaries + aliases), (2) sigil-prefixed live bindings (`%a`, `%all`, `%b`, `%parameters`, `@ARGV`, `@INC`, `@fpath`, `@path`, `$_`, `$stryke::VERSION`, …), (3) `CORE::`/`main::` qualified spellings of every callable / unqualified binding, plus stryke language keywords (`fn`, `match`, `class`, `struct`, `enum`, `mysync`, `frozen`, …). Drives the canonical `strykelang/lsp_completion_words.txt` snapshot consumed by both the static analyzer (linter) and LSP completion provider.

```
lsp_words |> ep                                          # browse everything
p len(lsp_words)                                          # ~18,000 entries
# Regenerate the on-disk snapshot:
# stryke -e 'lsp_words |> ep' > strykelang/lsp_completion_words.txt
```

lst_to_solar

`lst_to_solar` — batches 14-16 builtin.

```
my $result = lst_to_solar $input
```

lstat

`lstat` — filesystem builtin.

```
my $result = lstat $x
# or in a pipeline:
@list |> map lstat |> p
```

lucas

`lucas` — math / numeric (uncategorized batch) builtin.

```
my $result = lucas $x
# or in a pipeline:
@list |> map lucas |> p
```

luhn

`luhn` — math / numeric (uncategorized batch) builtin. Alias for `luhn_check`.

```
my $result = luhn $x
# or in a pipeline:
@list |> map luhn |> p
```

luhn_check

`luhn_check` — math / numeric (uncategorized batch) builtin.

```
my $result = luhn_check $x
# or in a pipeline:
@list |> map luhn_check |> p
```

luminance_relative

`luminance_relative` — batch 82 builtin.

```
my $result = luminance_relative $x
# or in a pipeline:
@list |> map luminance_relative |> p
```

lunation_index

`lunation_index` — batch 55 builtin.

```
my $result = lunation_index $x
# or in a pipeline:
@list |> map lunation_index |> p
```

lv_competition_step

`lv_competition_step` — batch 22 builtin.

```
my $result = lv_competition_step $x
# or in a pipeline:
@list |> map lv_competition_step |> p
```

ly_to_pc

`ly_to_pc` — batch 30 builtin.

```
my $result = ly_to_pc $input
```

lyndon_factorize

`lyndon_factorize` — batches 14-16 builtin.

```
my $result = lyndon_factorize $x
# or in a pipeline:
@list |> map lyndon_factorize |> p
```

lz

`lz` — base conversion builtin. Alias for `leading_zeros`.

```
my $result = lz $x
# or in a pipeline:
@list |> map lz |> p
```

lz4_block_step

`lz4_block_step` — batch 62 builtin.

```
my $result = lz4_block_step $x
# or in a pipeline:
@list |> map lz4_block_step |> p
```

lz4_encode_block

`lz4_encode_block` — batch 81 builtin.

```
my $result = lz4_encode_block $x
# or in a pipeline:
@list |> map lz4_encode_block |> p
```

lz4_match_offset

`lz4_match_offset` — batch 62 builtin.

```
my $result = lz4_match_offset $x
# or in a pipeline:
@list |> map lz4_match_offset |> p
```

lz77_match_length

`lz77_match_length` — batch 40 builtin.

```
my $result = lz77_match_length $x
# or in a pipeline:
@list |> map lz77_match_length |> p
```

lz78_dictionary_growth

`lz78_dictionary_growth` — batch 40 builtin.

```
my $result = lz78_dictionary_growth $x
# or in a pipeline:
@list |> map lz78_dictionary_growth |> p
```

lzma_encode_step

`lzma_encode_step` — batch 81 builtin.

```
my $result = lzma_encode_step $x
# or in a pipeline:
@list |> map lzma_encode_step |> p
```

lzma_range_step

`lzma_range_step` — batch 62 builtin.

```
my $result = lzma_range_step $x
# or in a pipeline:
@list |> map lzma_range_step |> p
```

lzo_encode_step

`lzo_encode_step` — batch 81 builtin.

```
my $result = lzo_encode_step $x
# or in a pipeline:
@list |> map lzo_encode_step |> p
```

lzw_encode

`lzw_encode` — batch 81 builtin.

```
my $result = lzw_encode $x
# or in a pipeline:
@list |> map lzw_encode |> p
```

lzw_step_dict

`lzw_step_dict` — batch 40 builtin.

```
my $result = lzw_step_dict $x
# or in a pipeline:
@list |> map lzw_step_dict |> p
```

m5

`m5` — crypto / encoding builtin. Alias for `md5`.

```
my $result = m5 $x
# or in a pipeline:
@list |> map m5 |> p
```

m_to_feet

`m_to_feet` — unit conversions builtin.

```
my $result = m_to_feet $input
```

m_to_miles

`m_to_miles` — unit conversions builtin.

```
my $result = m_to_miles $input
```

m_to_s

`m_to_s` — unit conversions builtin. Alias for `minutes_to_seconds`.

```
my $result = m_to_s $input
```

m_to_yards

`m_to_yards` — unit conversions builtin.

```
my $result = m_to_yards $input
```

ma_model_likelihood

`ma_model_likelihood` — batch 36 builtin.

```
my $result = ma_model_likelihood $x
# or in a pipeline:
@list |> map ma_model_likelihood |> p
```

mac_duration

`bond_duration` (alias `mac_duration`) computes Macaulay duration — the weighted-average time to receive cash flows.

```
my $dur = bond_duration([5,5,5,105], 0.05)  # ≈ 3.72 years
```

macarthur_wilson_extinction

`macarthur_wilson_extinction` — batch 22 builtin.

```
my $result = macarthur_wilson_extinction $x
# or in a pipeline:
@list |> map macarthur_wilson_extinction |> p
```

macarthur_wilson_immigration

`macarthur_wilson_immigration` — batch 22 builtin.

```
my $result = macarthur_wilson_immigration $x
# or in a pipeline:
@list |> map macarthur_wilson_immigration |> p
```

macaulay_duration

`macaulay_duration` — batch 20 builtin.

```
my $result = macaulay_duration $x
# or in a pipeline:
@list |> map macaulay_duration |> p
```

macauley_duration

`macauley_duration` — batch 57 builtin.

```
my $result = macauley_duration $x
# or in a pipeline:
@list |> map macauley_duration |> p
```

macdur

`macdur` — finance (extended) builtin. Alias for `duration`.

```
my $result = macdur $x
# or in a pipeline:
@list |> map macdur |> p
```

mach_full_step

`mach_full_step` — batch 42 builtin.

```
my $result = mach_full_step $x
# or in a pipeline:
@list |> map mach_full_step |> p
```

mad

`mad` — math / numeric (uncategorized batch) builtin. Alias for `median_absolute_deviation`.

```
my $result = mad $x
# or in a pipeline:
@list |> map mad |> p
```

madd

`madd` — extended stdlib builtin. Alias for `matrix_add`.

```
my $result = madd $x
# or in a pipeline:
@list |> map madd |> p
```

mae

`mae` — math functions builtin.

```
my $result = mae $x
# or in a pipeline:
@list |> map mae |> p
```

mag

`mag` — extended stdlib builtin. Alias for `magnitude`.

```
my $result = mag $x
# or in a pipeline:
@list |> map mag |> p
```

magenta

`magenta` — color / ansi builtin. Alias for `red`.

```
my $result = magenta $x
# or in a pipeline:
@list |> map magenta |> p
```

magenta_bold

`magenta_bold` — color / ansi builtin. Alias for `red`.

```
my $result = magenta_bold $x
# or in a pipeline:
@list |> map magenta_bold |> p
```

maglens

`maglens` — physics formulas builtin. Alias for `magnification_lens`.

```
my $result = maglens $x
# or in a pipeline:
@list |> map maglens |> p
```

magnification_lens

`magnification_lens` — physics formulas builtin.

```
my $result = magnification_lens $x
# or in a pipeline:
@list |> map magnification_lens |> p
```

magnitude

`magnitude` — extended stdlib builtin.

```
my $result = magnitude $x
# or in a pipeline:
@list |> map magnitude |> p
```

magnus_1st

`magnus_1st` — batch 34 builtin.

```
my $result = magnus_1st $x
# or in a pipeline:
@list |> map magnus_1st |> p
```

mahal

`mahalanobis` — Mahalanobis distance. Args: data, center, inverse_covariance. Like R's mahalanobis().

```
my @d = @{mahal([[1,2],[3,4]], [2,3], [[1,0],[0,1]])}
```

mahalanobis_1d

`mahalanobis_1d` — batch 27 builtin.

```
my $result = mahalanobis_1d $x
# or in a pipeline:
@list |> map mahalanobis_1d |> p
```

main

`main::` — the default package every script starts in. `our` variables and `fn` declarations without an explicit `package` directive land in `main::`; bare references like `$x` / `@a` / `%h` resolve to `$main::x` / `@main::a` / `%main::h` when looked up via the symbol table.

Use `main::` qualifier when you need to disambiguate inside a nested package, or when introspecting a script from a library context.

```
our $verbose = 1
p $main::verbose                            # 1 — same scalar

# Inside a class, the bare name resolves to the class — main:: gets you out:
class Foo {
    fn check { p $main::verbose }           # reads top-level our
}

# %main:: is the symbol-table stash. Lists every name declared
# at the top level (subs, our, our @, our %).
p scalar keys %main::                        # symbol-table size
keys %main:: |> sort |> p                   # browse every top-level name
```

Related namespaces: `CORE::` (every builtin), `stryke::` (reflection hashes — `%stryke::builtins` etc.), and `Pkg::` for any user-declared `package Pkg`.

main_sequence_lifetime

`main_sequence_lifetime` — batch 31 builtin.

```
my $result = main_sequence_lifetime $x
# or in a pipeline:
@list |> map main_sequence_lifetime |> p
```

mainardi_codazzi_step

`mainardi_codazzi_step` — batch 39 builtin.

```
my $result = mainardi_codazzi_step $x
# or in a pipeline:
@list |> map mainardi_codazzi_step |> p
```

majority

`majority` — extended stdlib batch 3 builtin. Alias for `majority_element`.

```
my $result = majority $x
# or in a pipeline:
@list |> map majority |> p
```

majority_element

`majority_element` — extended stdlib batch 3 builtin.

```
my $result = majority_element $x
# or in a pipeline:
@list |> map majority_element |> p
```

mall

`mall` — extended stdlib builtin. Alias for `match_all`.

```
my $result = mall $x
# or in a pipeline:
@list |> map mall |> p
```

mallat_pyramid_step

`mallat_pyramid_step` — batch 40 builtin.

```
my $result = mallat_pyramid_step $x
# or in a pipeline:
@list |> map mallat_pyramid_step |> p
```

manacher

`manacher` — batch 10 builtin. Alias for `manacher_radii`.

```
my $result = manacher $x
# or in a pipeline:
@list |> map manacher |> p
```

manacher_radii

`manacher_radii` — batch 10 builtin.

```
my $result = manacher_radii $x
# or in a pipeline:
@list |> map manacher_radii |> p
```

mandel

`mandel` — algorithms / puzzles builtin. Alias for `mandelbrot_char`.

```
my $result = mandel $x
# or in a pipeline:
@list |> map mandel |> p
```

mandel_q

`mandel_q` — batch 32 builtin.

```
my $result = mandel_q $x
# or in a pipeline:
@list |> map mandel_q |> p
```

mandelbrot_char

`mandelbrot_char` — algorithms / puzzles builtin.

```
my $result = mandelbrot_char $x
# or in a pipeline:
@list |> map mandelbrot_char |> p
```

manhattan_distance

`manhattan_distance` — extended stdlib builtin.

```
my $result = manhattan_distance $x
# or in a pipeline:
@list |> map manhattan_distance |> p
```

manipulability_yoshikawa

`manipulability_yoshikawa` — batch 59 builtin.

```
my $result = manipulability_yoshikawa $x
# or in a pipeline:
@list |> map manipulability_yoshikawa |> p
```

mann_kendall_tau

`mann_kendall_tau` — batches 14-16 builtin.

```
my $result = mann_kendall_tau $x
# or in a pipeline:
@list |> map mann_kendall_tau |> p
```

mann_whitney

`wilcox_test` (aliases `wilcox`, `mann_whitney`) — Wilcoxon rank-sum test (Mann-Whitney U). Returns U statistic. Like R's wilcox.test().

```
my $U = wilcox([1,2,3], [4,5,6])  # 0 (no overlap)
```

manning_velocity

`manning_velocity` — batch 13 builtin.

```
my $result = manning_velocity $x
# or in a pipeline:
@list |> map manning_velocity |> p
```

mannwhitneyu

`mannwhitneyu` — batch 53 builtin.

```
my $result = mannwhitneyu $x
# or in a pipeline:
@list |> map mannwhitneyu |> p
```

map_bp

`map_bp` — batch 12 builtin. Alias for `mean_arterial_pressure`.

```
my $result = map_bp $x
# or in a pipeline:
@list |> map map_bp |> p
```

map_chars

`map_chars` — string helpers (batch 4) builtin.

```
my $result = map_chars $x
# or in a pipeline:
@list |> map map_chars |> p
```

map_indexed

`map_indexed` — go/general functional utilities builtin.

```
my $result = map_indexed $x
# or in a pipeline:
@list |> map map_indexed |> p
```

map_keys_fn

`map_keys_fn` — hash ops (batch 2) builtin.

```
my $result = map_keys_fn $x
# or in a pipeline:
@list |> map map_keys_fn |> p
```

map_range

`map_range($value, $in_min, $in_max, $out_min, $out_max)` (alias `remap`) — remap a value from one range to another.

```
p remap(50, 0, 100, 0, 1)     # 0.5
p remap(75, 0, 100, 0, 255)   # 191.25 (percent to byte)
```

map_values_fn

`map_values_fn` — hash ops (batch 2) builtin.

```
my $result = map_values_fn $x
# or in a pipeline:
@list |> map map_values_fn |> p
```

map_while

`map_while` — rust iterator methods builtin.

```
my $result = map_while $x
# or in a pipeline:
@list |> map map_while |> p
```

mapcat

`mapcat` — algebraic match builtin.

```
my $result = mapcat $x
# or in a pipeline:
@list |> map mapcat |> p
```

mapi

`mapi` — go/general functional utilities builtin. Alias for `map_indexed`.

```
my $result = mapi $x
# or in a pipeline:
@list |> map mapi |> p
```

mapping_class_torus

`mapping_class_torus` — batch 37 builtin.

```
my $result = mapping_class_torus $x
# or in a pipeline:
@list |> map mapping_class_torus |> p
```

mapw

`mapw` — rust iterator methods builtin. Alias for `map_while`.

```
my $result = mapw $x
# or in a pipeline:
@list |> map mapw |> p
```

marcum_q

`marcum_q` — batch 72 builtin.

```
my $result = marcum_q $x
# or in a pipeline:
@list |> map marcum_q |> p
```

margalef_richness

`margalef_richness` — batch 22 builtin.

```
my $result = margalef_richness $x
# or in a pipeline:
@list |> map margalef_richness |> p
```

margin

`margin` — finance builtin.

```
my $result = margin $x
# or in a pipeline:
@list |> map margin |> p
```

margin_price

`margin_price` — physics formulas builtin.

```
my $result = margin_price $x
# or in a pipeline:
@list |> map margin_price |> p
```

margrabe

`margrabe` — batch 20 builtin.

```
my $result = margrabe $x
# or in a pipeline:
@list |> map margrabe |> p
```

markov_decision_value

`markov_decision_value` — batch 40 builtin.

```
my $result = markov_decision_value $x
# or in a pipeline:
@list |> map markov_decision_value |> p
```

markov_switching_ar

`markov_switching_ar` — batch 78 builtin.

```
my $result = markov_switching_ar $x
# or in a pipeline:
@list |> map markov_switching_ar |> p
```

markov_switching_mr

`markov_switching_mr` — batch 78 builtin.

```
my $result = markov_switching_mr $x
# or in a pipeline:
@list |> map markov_switching_mr |> p
```

markup

`markup` — finance builtin.

```
my $result = markup $x
# or in a pipeline:
@list |> map markup |> p
```

markup_price

`markup_price` — physics formulas builtin.

```
my $result = markup_price $x
# or in a pipeline:
@list |> map markup_price |> p
```

marshallian_demand

`marshallian_demand` — batch 73 builtin.

```
my $result = marshallian_demand $x
# or in a pipeline:
@list |> map marshallian_demand |> p
```

mask

`mask` — string processing (uncategorized batch) builtin. Alias for `mask_string`.

```
my $result = mask $x
# or in a pipeline:
@list |> map mask |> p
```

mask_string

`mask_string` — string processing (uncategorized batch) builtin.

```
my $result = mask_string $x
# or in a pipeline:
@list |> map mask_string |> p
```

mass_transport_overpotential

`mass_transport_overpotential` — batch 38 builtin.

```
my $result = mass_transport_overpotential $x
# or in a pipeline:
@list |> map mass_transport_overpotential |> p
```

mat_mul

`mat_mul` — matrix / linear algebra builtin. Alias for `matrix_multiply`.

```
my $result = mat_mul $x
# or in a pipeline:
@list |> map mat_mul |> p
```

mat_scale

`mat_scale` — extended stdlib builtin. Alias for `matrix_scale`.

```
my $result = mat_scale $x
# or in a pipeline:
@list |> map mat_scale |> p
```

match_all

`match_all` — extended stdlib builtin.

```
my $result = match_all $x
# or in a pipeline:
@list |> map match_all |> p
```

match_rating_codex

`match_rating_codex` — batch 69 builtin.

```
my $result = match_rating_codex $x
# or in a pipeline:
@list |> map match_rating_codex |> p
```

matches_regex

`matches_regex` — file stat / path builtin.

```
my $result = matches_regex $x
# or in a pipeline:
@list |> map matches_regex |> p
```

matching_market_step

`matching_market_step` — batch 43 builtin.

```
my $result = matching_market_step $x
# or in a pipeline:
@list |> map matching_market_step |> p
```

matching_pennies_payoff

`matching_pennies_payoff` — batch 43 builtin.

```
my $result = matching_pennies_payoff $x
# or in a pipeline:
@list |> map matching_pennies_payoff |> p
```

mathieu_b

`mathieu_b` — batch 72 builtin.

```
my $result = mathieu_b $x
# or in a pipeline:
@list |> map mathieu_b |> p
```

mathieu_ce0

`mathieu_ce0` — batch 29 builtin.

```
my $result = mathieu_ce0 $x
# or in a pipeline:
@list |> map mathieu_ce0 |> p
```

mathieu_se1

`mathieu_se1` — batch 29 builtin.

```
my $result = mathieu_se1 $x
# or in a pipeline:
@list |> map mathieu_se1 |> p
```

matmul

`matmul` — extended stdlib builtin. Alias for `matrix_mul`.

```
my $result = matmul $x
# or in a pipeline:
@list |> map matmul |> p
```

matrix_add

`matrix_add` — extended stdlib builtin.

```
my $result = matrix_add $x
# or in a pipeline:
@list |> map matrix_add |> p
```

matrix_anti_diagonal

`matrix_anti_diagonal` — batch 11 builtin.

```
my $result = matrix_anti_diagonal $x
# or in a pipeline:
@list |> map matrix_anti_diagonal |> p
```

matrix_col

`matrix_col` — extended stdlib builtin.

```
my $result = matrix_col $x
# or in a pipeline:
@list |> map matrix_col |> p
```

matrix_det

`matrix_det` — extended stdlib builtin.

```
my $result = matrix_det $x
# or in a pipeline:
@list |> map matrix_det |> p
```

matrix_diag

`matrix_diag` — extended stdlib builtin.

```
my $result = matrix_diag $x
# or in a pipeline:
@list |> map matrix_diag |> p
```

matrix_diagonal

`matrix_diagonal` — batch 11 builtin.

```
my $result = matrix_diagonal $x
# or in a pipeline:
@list |> map matrix_diagonal |> p
```

matrix_flatten

`matrix_flatten` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_flatten $x
# or in a pipeline:
@list |> map matrix_flatten |> p
```

matrix_from_rows

`matrix_from_rows` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_from_rows $x
# or in a pipeline:
@list |> map matrix_from_rows |> p
```

matrix_hadamard

`matrix_hadamard` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_hadamard $x
# or in a pipeline:
@list |> map matrix_hadamard |> p
```

matrix_identity

`matrix_identity` — extended stdlib builtin.

```
my $result = matrix_identity $x
# or in a pipeline:
@list |> map matrix_identity |> p
```

matrix_identity_q

`matrix_identity_q` — batches 17-18 builtin.

```
my $result = matrix_identity_q $x
# or in a pipeline:
@list |> map matrix_identity_q |> p
```

matrix_inverse

`matrix_inverse` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_inverse $x
# or in a pipeline:
@list |> map matrix_inverse |> p
```

matrix_lu

`matrix_lu` (alias `mlu`) computes the LU decomposition with partial pivoting. Returns [L, U, P] where PA = LU.

```
my ($L, $U, $P) = @{mlu([[4,3],[6,3]])}
```

matrix_map

`matrix_map` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_map $x
# or in a pipeline:
@list |> map matrix_map |> p
```

matrix_max

`matrix_max` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_max $x
# or in a pipeline:
@list |> map matrix_max |> p
```

matrix_min

`matrix_min` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_min $x
# or in a pipeline:
@list |> map matrix_min |> p
```

matrix_mul

`matrix_mul` — extended stdlib builtin.

```
my $result = matrix_mul $x
# or in a pipeline:
@list |> map matrix_mul |> p
```

matrix_mult

`matrix_mult` — extended stdlib builtin.

```
my $result = matrix_mult $x
# or in a pipeline:
@list |> map matrix_mult |> p
```

matrix_multiply

`matrix_multiply` — matrix / linear algebra builtin.

```
my $result = matrix_multiply $x
# or in a pipeline:
@list |> map matrix_multiply |> p
```

matrix_norm

`matrix_norm` (alias `mnorm`) computes a matrix norm. Default is Frobenius; pass 1 for max-column-sum, Inf for max-row-sum.

```
p mnorm([[3,4]])           # 5 (Frobenius)
p mnorm([[1,2],[3,4]], 1)  # 6 (1-norm)
```

matrix_ones

`matrix_ones` — extended stdlib builtin.

```
my $result = matrix_ones $x
# or in a pipeline:
@list |> map matrix_ones |> p
```

matrix_orthogonal_q

`matrix_orthogonal_q` — batch 11 builtin.

```
my $result = matrix_orthogonal_q $x
# or in a pipeline:
@list |> map matrix_orthogonal_q |> p
```

matrix_pinv

`matrix_pinv` (aliases `mpinv`, `pinv`) computes the Moore-Penrose pseudo-inverse via (A^T A)^{-1} A^T.

```
my $A_plus = pinv([[1,2],[3,4],[5,6]])
```

matrix_power

`matrix_power` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_power $x
# or in a pipeline:
@list |> map matrix_power |> p
```

matrix_qr

`matrix_qr` (alias `mqr`) computes the QR decomposition via Gram-Schmidt orthogonalization. Returns [Q, R] where A = QR.

```
my ($Q, $R) = @{mqr([[1,1],[1,-1]])}
```

matrix_random_normal

`matrix_random_normal` — batches 17-18 builtin.

```
my $result = matrix_random_normal $x
# or in a pipeline:
@list |> map matrix_random_normal |> p
```

matrix_random_uniform

`matrix_random_uniform` — batches 17-18 builtin.

```
my $result = matrix_random_uniform $x
# or in a pipeline:
@list |> map matrix_random_uniform |> p
```

matrix_row

`matrix_row` — extended stdlib builtin.

```
my $result = matrix_row $x
# or in a pipeline:
@list |> map matrix_row |> p
```

matrix_scalar

`matrix_scalar` — extended stdlib builtin.

```
my $result = matrix_scalar $x
# or in a pipeline:
@list |> map matrix_scalar |> p
```

matrix_scale

`matrix_scale` — extended stdlib builtin.

```
my $result = matrix_scale $x
# or in a pipeline:
@list |> map matrix_scale |> p
```

matrix_shape

`matrix_shape` — extended stdlib builtin.

```
my $result = matrix_shape $x
# or in a pipeline:
@list |> map matrix_shape |> p
```

matrix_solve

`matrix_solve` (aliases `msolve`, `solve`) solves the linear system Ax=b via Gaussian elimination with partial pivoting. Returns the solution vector x.

```
my $A = [[2,1],[-1,1]]
my $b = [5,2]
my $x = solve($A, $b)   # [1, 3]
```

matrix_sub

`matrix_sub` — extended stdlib builtin.

```
my $result = matrix_sub $x
# or in a pipeline:
@list |> map matrix_sub |> p
```

matrix_sum

`matrix_sum` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_sum $x
# or in a pipeline:
@list |> map matrix_sum |> p
```

matrix_symmetric_q

`matrix_symmetric_q` — batch 11 builtin.

```
my $result = matrix_symmetric_q $x
# or in a pipeline:
@list |> map matrix_symmetric_q |> p
```

matrix_trace

`matrix_trace` — extended stdlib builtin.

```
my $result = matrix_trace $x
# or in a pipeline:
@list |> map matrix_trace |> p
```

matrix_transpose

`matrix_transpose` — matrix operations (uncategorized batch) builtin.

```
my $result = matrix_transpose $x
# or in a pipeline:
@list |> map matrix_transpose |> p
```

matrix_zero_q

`matrix_zero_q` — batches 17-18 builtin.

```
my $result = matrix_zero_q $x
# or in a pipeline:
@list |> map matrix_zero_q |> p
```

matrix_zeros

`matrix_zeros` — extended stdlib builtin.

```
my $result = matrix_zeros $x
# or in a pipeline:
@list |> map matrix_zeros |> p
```

matroid_greedy_step

`matroid_greedy_step` — batch 41 builtin.

```
my $result = matroid_greedy_step $x
# or in a pipeline:
@list |> map matroid_greedy_step |> p
```

matroid_intersection_step

`matroid_intersection_step` — batch 41 builtin.

```
my $result = matroid_intersection_step $x
# or in a pipeline:
@list |> map matroid_intersection_step |> p
```

max2

`max2` — trivial numeric / predicate builtins builtin.

```
my $result = max2 $x
# or in a pipeline:
@list |> map max2 |> p
```

max_abs

`max_abs` — trig / math (batch 2) builtin.

```
my $result = max_abs $x
# or in a pipeline:
@list |> map max_abs |> p
```

max_by

`max_by` — functional / iterator builtin. Alias for `take_while`.

```
my $result = max_by $x
# or in a pipeline:
@list |> map max_by |> p
```

max_degree

`max_degree` — batch 24 builtin.

```
my $result = max_degree $x
# or in a pipeline:
@list |> map max_degree |> p
```

max_drawdown

`max_drawdown(@equity_curve)` (alias `mdd`) — computes the maximum peak-to-trough decline in a series of values. Returns as a decimal (0.2 = 20% drawdown). Key risk metric.

```
my @equity = (100, 120, 90, 110, 85, 130)
p mdd(@equity)   # ~0.29 (29% max drawdown)
```

max_drawdown_arr

`max_drawdown_arr` — batches 17-18 builtin.

```
my $result = max_drawdown_arr $x
# or in a pipeline:
@list |> map max_drawdown_arr |> p
```

max_float

`max_float` — conversion / utility (batch 4) builtin.

```
my $result = max_float $x
# or in a pipeline:
@list |> map max_float |> p
```

max_flow_ek

`max_flow_ek` — batch 24 builtin.

```
my $result = max_flow_ek $x
# or in a pipeline:
@list |> map max_flow_ek |> p
```

max_flow_ford_fulkerson_step

`max_flow_ford_fulkerson_step` — batch 41 builtin.

```
my $result = max_flow_ford_fulkerson_step $x
# or in a pipeline:
@list |> map max_flow_ford_fulkerson_step |> p
```

max_heart_rate

`max_heart_rate` — batch 12 builtin.

```
my $result = max_heart_rate $x
# or in a pipeline:
@list |> map max_heart_rate |> p
```

max_independent_set

`max_independent_set` — batch 75 builtin.

```
my $result = max_independent_set $x
# or in a pipeline:
@list |> map max_independent_set |> p
```

max_list

`max_list` — list helpers (batch 4) builtin.

```
my $result = max_list $x
# or in a pipeline:
@list |> map max_list |> p
```

max_norm

`max_norm` — batch 11 builtin. Alias for `linf_norm`.

```
my $result = max_norm $x
# or in a pipeline:
@list |> map max_norm |> p
```

max_of

`max_of` — more list helpers builtin.

```
my $result = max_of $x
# or in a pipeline:
@list |> map max_of |> p
```

max_sum_subarray

`max_sum_subarray` — extended stdlib batch 3 builtin.

```
my $result = max_sum_subarray $x
# or in a pipeline:
@list |> map max_sum_subarray |> p
```

maxby

`maxby` — additional missing stdlib functions builtin. Alias for `maximum_by`.

```
my $result = maxby $x
# or in a pipeline:
@list |> map maxby |> p
```

maxdd

`maxdd` — finance (extended) builtin. Alias for `max_drawdown`.

```
my $result = maxdd $x
# or in a pipeline:
@list |> map maxdd |> p
```

maximin_share

`maximin_share` — batch 43 builtin.

```
my $result = maximin_share $x
# or in a pipeline:
@list |> map maximin_share |> p
```

maximum_bipartite_matching

`maximum_bipartite_matching` — batch 41 builtin.

```
my $result = maximum_bipartite_matching $x
# or in a pipeline:
@list |> map maximum_bipartite_matching |> p
```

maximum_by

`maximum_by` — additional missing stdlib functions builtin.

```
my $result = maximum_by $x
# or in a pipeline:
@list |> map maximum_by |> p
```

maxstr

`maxstr` — list / aggregate builtin.

```
my $result = maxstr $x
# or in a pipeline:
@list |> map maxstr |> p
```

maxsub

`maxsub` — extended stdlib batch 3 builtin. Alias for `max_sum_subarray`.

```
my $result = maxsub $x
# or in a pipeline:
@list |> map maxsub |> p
```

maya_long_count_from_fixed

`maya_long_count_from_fixed` — batch 52 builtin.

```
my $result = maya_long_count_from_fixed $x
# or in a pipeline:
@list |> map maya_long_count_from_fixed |> p
```

mayan_haab_from_fixed

`mayan_haab_from_fixed` — batch 52 builtin.

```
my $result = mayan_haab_from_fixed $x
# or in a pipeline:
@list |> map mayan_haab_from_fixed |> p
```

mayan_tzolkin_from_fixed

`mayan_tzolkin_from_fixed` — batch 52 builtin.

```
my $result = mayan_tzolkin_from_fixed $x
# or in a pipeline:
@list |> map mayan_tzolkin_from_fixed |> p
```

mb_to_bytes

`mb_to_bytes` — unit conversions builtin.

```
my $result = mb_to_bytes $input
```

mb_to_gb

`mb_to_gb` — unit conversions builtin.

```
my $result = mb_to_gb $input
```

mc_limiter

`mc_limiter` — batch 34 builtin.

```
my $result = mc_limiter $x
# or in a pipeline:
@list |> map mc_limiter |> p
```

mcat

`mcat` — algebraic match builtin. Alias for `mapcat`.

```
my $result = mcat $x
# or in a pipeline:
@list |> map mcat |> p
```

mcc

`mcc` — batch 27 builtin.

```
my $result = mcc $x
# or in a pipeline:
@list |> map mcc |> p
```

mcol

`mcol` — extended stdlib builtin. Alias for `matrix_col`.

```
my $result = mcol $x
# or in a pipeline:
@list |> map mcol |> p
```

mcp_serve_registered_tools

`mcp_serve_registered_tools("server-name")` — convert every tool registered via `tool fn` / `ai_register_tool` into MCP server schema and run the stdio JSON-RPC loop. Used by `stryke build --mcp-server`.

md5

`md5` — crypto / encoding builtin.

```
my $result = md5 $x
# or in a pipeline:
@list |> map md5 |> p
```

mdet

`mdet` — extended stdlib builtin. Alias for `matrix_det`.

```
my $result = mdet $x
# or in a pipeline:
@list |> map mdet |> p
```

mdiag

`mdiag` — extended stdlib builtin. Alias for `matrix_diag`.

```
my $result = mdiag $x
# or in a pipeline:
@list |> map mdiag |> p
```

mdiagvec

`mdiagvec` — batch 11 builtin. Alias for `matrix_diagonal`.

```
my $result = mdiagvec $x
# or in a pipeline:
@list |> map mdiagvec |> p
```

mdist

`mdist` — extended stdlib builtin. Alias for `manhattan_distance`.

```
my $result = mdist $x
# or in a pipeline:
@list |> map mdist |> p
```

mduration

`mduration` — batch 81 builtin.

```
my $result = mduration $x
# or in a pipeline:
@list |> map mduration |> p
```

mean_absolute_error

`mean_absolute_error` — math / numeric (uncategorized batch) builtin.

```
my $result = mean_absolute_error $x
# or in a pipeline:
@list |> map mean_absolute_error |> p
```

mean_activity_coeff_pitzer

`mean_activity_coeff_pitzer` — batch 38 builtin.

```
my $result = mean_activity_coeff_pitzer $x
# or in a pipeline:
@list |> map mean_activity_coeff_pitzer |> p
```

mean_arterial_pressure

`mean_arterial_pressure` — batch 12 builtin.

```
my $result = mean_arterial_pressure $x
# or in a pipeline:
@list |> map mean_arterial_pressure |> p
```

mean_curvature_step

`mean_curvature_step` — batch 39 builtin.

```
my $result = mean_curvature_step $x
# or in a pipeline:
@list |> map mean_curvature_step |> p
```

mean_filter_window

`mean_filter_window` — batch 40 builtin.

```
my $result = mean_filter_window $x
# or in a pipeline:
@list |> map mean_filter_window |> p
```

mean_list

`mean_list` — list helpers (batch 4) builtin.

```
my $result = mean_list $x
# or in a pipeline:
@list |> map mean_list |> p
```

mean_solar_anomaly

`mean_solar_anomaly` — batches 14-16 builtin.

```
my $result = mean_solar_anomaly $x
# or in a pipeline:
@list |> map mean_solar_anomaly |> p
```

mean_solar_longitude

`mean_solar_longitude` — batches 14-16 builtin.

```
my $result = mean_solar_longitude $x
# or in a pipeline:
@list |> map mean_solar_longitude |> p
```

mean_squared_error

`mean_squared_error` — math / numeric (uncategorized batch) builtin.

```
my $result = mean_squared_error $x
# or in a pipeline:
@list |> map mean_squared_error |> p
```

mean_tone_freq

`mean_tone_freq` — batch 64 builtin.

```
my $result = mean_tone_freq $x
# or in a pipeline:
@list |> map mean_tone_freq |> p
```

measure

`measure` — conversion / utility (batch 4) builtin.

```
my $result = measure $x
# or in a pipeline:
@list |> map measure |> p
```

mechanism_design_obj

`mechanism_design_obj` — batch 43 builtin.

```
my $result = mechanism_design_obj $x
# or in a pipeline:
@list |> map mechanism_design_obj |> p
```

medfilt

`median_filter(\@signal, $window_size)` (alias `medfilt`) — applies a median filter for noise removal. Each output is the median of the surrounding window. Excellent for removing impulse noise while preserving edges.

```
my @spiky = (1, 2, 100, 3, 4)   # 100 is spike
my $clean = medfilt(\@spiky, 3)
p @$clean   # spike removed
```

medfilt_1d

`medfilt_1d` — batch 74 builtin.

```
my $result = medfilt_1d $x
# or in a pipeline:
@list |> map medfilt_1d |> p
```

median_absolute_deviation

`median_absolute_deviation` — math / numeric (uncategorized batch) builtin.

```
my $result = median_absolute_deviation $x
# or in a pipeline:
@list |> map median_absolute_deviation |> p
```

median_filter_window

median_filter_window — batch 40 builtin.

```
my $result = median_filter_window $x
# or in a pipeline:
@list |> map median_filter_window |> p
```

median_voter_step

median_voter_step — batch 43 builtin.

```
my $result = median_voter_step $x
# or in a pipeline:
@list |> map median_voter_step |> p
```

mem_free

mem_free — free/available physical RAM in bytes. On Linux reads MemAvailable from /proc/meminfo (falls back to MemFree). On macOS uses vm.page_free_count via sysctl. Returns undef on unsupported platforms.

```
p mem_free                          # 3821666304
p format_bytes(mem_free)            # 3.6 GiB
my $pct = int(mem_free / mem_total * 100)
p "$pct% free"
```

mem_total

mem_total — total physical RAM in bytes. Uses /proc/meminfo on Linux, sysctlbyname("hw.memsize") on macOS. Returns undef on unsupported platforms.

```
p mem_total                         # 68719476736
p format_bytes(mem_total)           # 64.0 GiB
```

mem_used

mem_used — used physical RAM in bytes (total - free). Returns undef if either mem_total or mem_free is unavailable.

```
p format_bytes(mem_used)            # 60.4 GiB
gauge(mem_used / mem_total) |> p    # memory usage gauge
```

membership_idx

membership_idx — batch 54 builtin.

```
my $result = membership_idx $x
# or in a pipeline:
@list |> map membership_idx |> p
```

menhinick_richness

menhinick_richness — batch 22 builtin.

```
my $result = menhinick_richness $x
# or in a pipeline:
@list |> map menhinick_richness |> p
```

mercury_perihelion_advance

`mercury_perihelion_advance` — batch 39 builtin.

```
my $result = mercury_perihelion_advance $x
# or in a pipeline:
@list |> map mercury_perihelion_advance |> p
```

merge_hash

`merge_hash` — hash helpers builtin.

```
my $result = merge_hash $x
# or in a pipeline:
@list |> map merge_hash |> p
```

merge_sorted

`merge_sorted` — matrix operations (uncategorized batch) builtin.

```
my $result = merge_sorted $x
# or in a pipeline:
@list |> map merge_sorted |> p
```

mertens

`mertens` — batch 35 builtin.

```
my $result = mertens $x
# or in a pipeline:
@list |> map mertens |> p
```

mertens_zamir_step

`mertens_zamir_step` — batch 43 builtin.

```
my $result = mertens_zamir_step $x
# or in a pipeline:
@list |> map mertens_zamir_step |> p
```

merton_jump_call

`merton_jump_call` — batch 20 builtin.

```
my $result = merton_jump_call $x
# or in a pipeline:
@list |> map merton_jump_call |> p
```

metal_corrosion_rate

`metal_corrosion_rate` — batch 38 builtin.

```
my $result = metal_corrosion_rate $x
# or in a pipeline:
@list |> map metal_corrosion_rate |> p
```

metamorphic_grade

`metamorphic_grade` — batch 65 builtin.

```
my $result = metamorphic_grade $x
# or in a pipeline:
@list |> map metamorphic_grade |> p
```

metaphone

`metaphone` — encoding / phonetics builtin.

```
my $result = metaphone $x
# or in a pipeline:
@list |> map metaphone |> p
```

metaphone_phonetic

`metaphone_phonetic` — batch 69 builtin.

```
my $result = metaphone_phonetic $x
# or in a pipeline:
@list |> map metaphone_phonetic |> p
```

meters_to_feet

`meters_to_feet` — batch 30 builtin.

```
my $result = meters_to_feet $input
```

methane_growth_rate

`methane_growth_rate` — batch 42 builtin.

```
my $result = methane_growth_rate $x
# or in a pipeline:
@list |> map methane_growth_rate |> p
```

metonic_cycle

`metonic_cycle` — batch 55 builtin.

```
my $result = metonic_cycle $x
# or in a pipeline:
@list |> map metonic_cycle |> p
```

metph

`metph` — encoding / phonetics builtin. Alias for `metaphone`.

```
my $result = metph $x
# or in a pipeline:
@list |> map metph |> p
```

metric_frw_lapse

`metric_frw_lapse` — batch 39 builtin.

```
my $result = metric_frw_lapse $x
# or in a pipeline:
@list |> map metric_frw_lapse |> p
```

metric_kerr_step_simple

`metric_kerr_step_simple` — batch 39 builtin.

```
my $result = metric_kerr_step_simple $x
# or in a pipeline:
@list |> map metric_kerr_step_simple |> p
```

metric_minkowski_eta_step

`metric_minkowski_eta_step` — batch 39 builtin.

```
my $result = metric_minkowski_eta_step $x
# or in a pipeline:
@list |> map metric_minkowski_eta_step |> p
```

metric_schwarzschild_step

`metric_schwarzschild_step` — batch 39 builtin.

```
my $result = metric_schwarzschild_step $x
# or in a pipeline:
@list |> map metric_schwarzschild_step |> p
```

metro_hash_64

`metro_hash_64` — batch 81 builtin.

```
my $result = metro_hash_64 $x
# or in a pipeline:
@list |> map metro_hash_64 |> p
```

mexican_hat_wavelet

`mexican_hat_wavelet` — batch 74 builtin.

```
my $result = mexican_hat_wavelet $x
# or in a pipeline:
@list |> map mexican_hat_wavelet |> p
```

mfcc_coeff_step

`mfcc_coeff_step` — batch 40 builtin.

```
my $result = mfcc_coeff_step $x
# or in a pipeline:
@list |> map mfcc_coeff_step |> p
```

mflat

`mflat` — matrix operations (uncategorized batch) builtin. Alias for `matrix_flatten`.

```
my $result = mflat $x
# or in a pipeline:
@list |> map mflat |> p
```

mfromr

`mfromr` — matrix operations (uncategorized batch) builtin. Alias for `matrix_from_rows`.

```
my $result = mfromr $x
# or in a pipeline:
@list |> map mfromr |> p
```

mget

`mget` — batch 71 builtin.

```
my $result = mget $x
# or in a pipeline:
@list |> map mget |> p
```

mgrs_decode

`mgrs_decode` — batch 58 builtin.

```
my $result = mgrs_decode $x
# or in a pipeline:
@list |> map mgrs_decode |> p
```

mgrs_encode

`mgrs_encode` — batch 58 builtin.

```
my $result = mgrs_encode $x
# or in a pipeline:
@list |> map mgrs_encode |> p
```

mhad

`mhad` — matrix operations (uncategorized batch) builtin. Alias for `matrix_hadamard`.

```
my $result = mhad $x
# or in a pipeline:
@list |> map mhad |> p
```

michaelis_menten

`michaelis_menten` — batch 12 builtin.

```
my $result = michaelis_menten $x
# or in a pipeline:
@list |> map michaelis_menten |> p
```

microlensing_magnification

`microlensing_magnification` — batch 31 builtin.

```
my $result = microlensing_magnification $x
# or in a pipeline:
@list |> map microlensing_magnification |> p
```

mid_str

`mid_str` — string (batch 2) builtin.

```
my $result = mid_str $x
# or in a pipeline:
@list |> map mid_str |> p
```

mident

`mident` — extended stdlib builtin. Alias for `matrix_identity`.

```
my $result = mident $x
# or in a pipeline:
@list |> map mident |> p
```

midi_to_hz

`midi_to_hz` — batch 30 builtin.

```
my $result = midi_to_hz $input
```

midi_to_pitch_class

`midi_to_pitch_class` — batch 9 builtin.

```
my $result = midi_to_pitch_class $input
```

midpnt

`midpnt` — geometry / physics builtin. Alias for `midpoint`.

```
my $result = midpnt $x
# or in a pipeline:
@list |> map midpnt |> p
```

midpoint

`midpoint` — geometry / physics builtin.

```
my $result = midpoint $x
# or in a pipeline:
@list |> map midpoint |> p
```

midpoint_great_circle

`midpoint_great_circle` — batch 28 builtin.

```
my $result = midpoint_great_circle $x
# or in a pipeline:
@list |> map midpoint_great_circle |> p
```

midpoint_of

`midpoint_of` — extended stdlib builtin.

```
my $result = midpoint_of $x
# or in a pipeline:
@list |> map midpoint_of |> p
```

midpoint_rule

`midpoint_rule` — batch 19 builtin.

```
my $result = midpoint_rule $x
# or in a pipeline:
@list |> map midpoint_rule |> p
```

midpt

`midpt` — extended stdlib builtin. Alias for `midpoint_of`.

```
my $result = midpt $x
# or in a pipeline:
@list |> map midpt |> p
```

miles_to_km

`miles_to_km` — unit conversions builtin.

```
my $result = miles_to_km $input
```

miles_to_m

`miles_to_m` — unit conversions builtin.

```
my $result = miles_to_m $input
```

milne

`milne` — batch 19 builtin. Alias for `milne_pc`.

```
my $result = milne $x
# or in a pipeline:
@list |> map milne |> p
```

milne_pc

`milne_pc` — batch 19 builtin.

```
my $result = milne_pc $x
# or in a pipeline:
@list |> map milne_pc |> p
```

milstein_step

`milstein_step` — batch 34 builtin.

```
my $result = milstein_step $x
# or in a pipeline:
@list |> map milstein_step |> p
```

min2

`min2` — trivial numeric / predicate builtins builtin.

```
my $result = min2 $x
# or in a pipeline:
@list |> map min2 |> p
```

min_abs

`min_abs` — trig / math (batch 2) builtin.

```
my $result = min_abs $x
# or in a pipeline:
@list |> map min_abs |> p
```

min_by

`min_by` — functional / iterator builtin. Alias for `take_while`.

```
my $result = min_by $x
# or in a pipeline:
@list |> map min_by |> p
```

min_cut_value

`min_cut_value` — batch 24 builtin.

```
my $result = min_cut_value $x
# or in a pipeline:
@list |> map min_cut_value |> p
```

min_float

`min_float` — conversion / utility (batch 4) builtin.

```
my $result = min_float $x
# or in a pipeline:
@list |> map min_float |> p
```

min_list

`min_list` — list helpers (batch 4) builtin.

```
my $result = min_list $x
# or in a pipeline:
@list |> map min_list |> p
```

min_max

`min_max` — file stat / path builtin.

```
my $result = min_max $x
# or in a pipeline:
@list |> map min_max |> p
```

min_of

`min_of` — more list helpers builtin.

```
my $result = min_of $x
# or in a pipeline:
@list |> map min_of |> p
```

min_steiner_tree

`min_steiner_tree` — batch 75 builtin.

```
my $result = min_steiner_tree $x
# or in a pipeline:
@list |> map min_steiner_tree |> p
```

min_vertex_cover

`min_vertex_cover` — batch 75 builtin.

```
my $result = min_vertex_cover $x
# or in a pipeline:
@list |> map min_vertex_cover |> p
```

minby

`minby` — additional missing stdlib functions builtin. Alias for `minimum_by`.

```
my $result = minby $x
# or in a pipeline:
@list |> map minby |> p
```

mincut_stoer_wagner

`mincut_stoer_wagner` — batch 41 builtin.

```
my $result = mincut_stoer_wagner $x
# or in a pipeline:
@list |> map mincut_stoer_wagner |> p
```

mineral_mohs_hardness

`mineral_mohs_hardness` — batch 65 builtin.

```
my $result = mineral_mohs_hardness $x
# or in a pipeline:
@list |> map mineral_mohs_hardness |> p
```

minimum_by

`minimum_by` — additional missing stdlib functions builtin.

```
my $result = minimum_by $x
# or in a pipeline:
@list |> map minimum_by |> p
```

minkdist

`minkdist` — math / numeric (uncategorized batch) builtin. Alias for `minkowski_distance`.

```
my $result = minkdist $x
# or in a pipeline:
@list |> map minkdist |> p
```

minkowski_distance

`minkowski_distance` — math / numeric (uncategorized batch) builtin.

```
my $result = minkowski_distance $x
# or in a pipeline:
@list |> map minkowski_distance |> p
```

minkowski_sum_simple

`minkowski_sum_simple` — batches 17-18 builtin.

```
my $result = minkowski_sum_simple $x
# or in a pipeline:
@list |> map minkowski_sum_simple |> p
```

minmax

`minmax` — python/ruby stdlib builtin.

```
my $result = minmax $x
# or in a pipeline:
@list |> map minmax |> p
```

minmax_by

`minmax_by` — python/ruby stdlib builtin.

```
my $result = minmax_by $x
# or in a pipeline:
@list |> map minmax_by |> p
```

minmax_scale

`minmax_scale` — batch 27 builtin.

```
my $result = minmax_scale $x
# or in a pipeline:
@list |> map minmax_scale |> p
```

minmod_limiter

`minmod_limiter` — batch 34 builtin.

```
my $result = minmod_limiter $x
# or in a pipeline:
@list |> map minmod_limiter |> p
```

minstr

`minstr` — list / aggregate builtin.

```
my $result = minstr $x
# or in a pipeline:
@list |> map minstr |> p
```

minutes_to_hours

`minutes_to_hours` — unit conversions builtin.

```
my $result = minutes_to_hours $input
```

minutes_to_seconds

`minutes_to_seconds` — unit conversions builtin.

```
my $result = minutes_to_seconds $input
```

minv

`minv` — matrix operations (uncategorized batch) builtin. Alias for `matrix_inverse`.

```
my $result = minv $x
# or in a pipeline:
@list |> map minv |> p
```

mirr

`mirr` — batch 10 builtin. Alias for `modified_irr`.

```
my $result = mirr $x
# or in a pipeline:
@list |> map mirr |> p
```

mirr_excel

`mirr_excel` — batch 81 builtin.

```
my $result = mirr_excel $x
# or in a pipeline:
@list |> map mirr_excel |> p
```

mirror_descent_step

`mirror_descent_step` — batch 41 builtin.

```
my $result = mirror_descent_step $x
# or in a pipeline:
@list |> map mirror_descent_step |> p
```

mirror_equation_v

`mirror_equation_v` — batch 23 builtin.

```
my $result = mirror_equation_v $x
# or in a pipeline:
@list |> map mirror_equation_v |> p
```

mirror_string

`mirror_string` — string helpers (batch 4) builtin.

```
my $result = mirror_string $x
# or in a pipeline:
@list |> map mirror_string |> p
```

mirror_symmetry_check

`mirror_symmetry_check` — batch 37 builtin.

```
my $result = mirror_symmetry_check $x
# or in a pipeline:
@list |> map mirror_symmetry_check |> p
```

mish

`mish` applies the Mish activation: x·tanh(softplus(x)). Often outperforms ReLU/Swish.

```
p mish(1)    # 0.8651
p mish(-1)   # -0.3034
```

mixed_integer_round_down

`mixed_integer_round_down` — batch 41 builtin.

```
my $result = mixed_integer_round_down $x
# or in a pipeline:
@list |> map mixed_integer_round_down |> p
```

mixed_integer_round_up

`mixed_integer_round_up` — batch 41 builtin.

```
my $result = mixed_integer_round_up $x
# or in a pipeline:
@list |> map mixed_integer_round_up |> p
```

mixed_potential_step

mixed_potential_step — batch 38 builtin.

```
my $result = mixed_potential_step $x
# or in a pipeline:
@list |> map mixed_potential_step |> p
```

mixed_strategy_value

mixed_strategy_value — batch 43 builtin.

```
my $result = mixed_strategy_value $x
# or in a pipeline:
@list |> map mixed_strategy_value |> p
```

mixing_length_prandtl

mixing_length_prandtl — batch 42 builtin.

```
my $result = mixing_length_prandtl $x
# or in a pipeline:
@list |> map mixing_length_prandtl |> p
```

mjo_phase_step

mjo_phase_step — batch 42 builtin.

```
my $result = mjo_phase_step $x
# or in a pipeline:
@list |> map mjo_phase_step |> p
```

ml_adabelief_step

ml_adabelief_step — batch 45 builtin.

```
my $result = ml_adabelief_step $x
# or in a pipeline:
@list |> map ml_adabelief_step |> p
```

ml_adagrad_step

ml_adagrad_step — batch 45 builtin.

```
my $result = ml_adagrad_step $x
# or in a pipeline:
@list |> map ml_adagrad_step |> p
```

ml_adam_step

ml_adam_step — batch 45 builtin.

```
my $result = ml_adam_step $x
# or in a pipeline:
@list |> map ml_adam_step |> p
```

ml_adamax_step

ml_adamax_step — batch 45 builtin.

```
my $result = ml_adamax_step $x
# or in a pipeline:
@list |> map ml_adamax_step |> p
```

ml_adamw_step

`ml_adamw_step` — batch 45 builtin.

```
my $result = ml_adamw_step $x
# or in a pipeline:
@list |> map ml_adamw_step |> p
```

ml_amsgrad_step

`ml_amsgrad_step` — batch 45 builtin.

```
my $result = ml_amsgrad_step $x
# or in a pipeline:
@list |> map ml_amsgrad_step |> p
```

ml_arcface_loss_step

`ml_arcface_loss_step` — batch 45 builtin.

```
my $result = ml_arcface_loss_step $x
# or in a pipeline:
@list |> map ml_arcface_loss_step |> p
```

ml_attention_score_step

`ml_attention_score_step` — batch 45 builtin.

```
my $result = ml_attention_score_step $x
# or in a pipeline:
@list |> map ml_attention_score_step |> p
```

ml_batch_norm_step

`ml_batch_norm_step` — batch 45 builtin.

```
my $result = ml_batch_norm_step $x
# or in a pipeline:
@list |> map ml_batch_norm_step |> p
```

ml_binary_cross_entropy

`ml_binary_cross_entropy` — batch 45 builtin.

```
my $result = ml_binary_cross_entropy $x
# or in a pipeline:
@list |> map ml_binary_cross_entropy |> p
```

ml_celu_step

`ml_celu_step` — batch 45 builtin.

```
my $result = ml_celu_step $x
# or in a pipeline:
@list |> map ml_celu_step |> p
```

ml_center_loss_step

`ml_center_loss_step` — batch 45 builtin.

```
my $result = ml_center_loss_step $x
# or in a pipeline:
@list |> map ml_center_loss_step |> p
```

ml_ciou_loss_step

`ml_ciou_loss_step` — batch 45 builtin.

```
my $result = ml_ciou_loss_step $x
# or in a pipeline:
@list |> map ml_ciou_loss_step |> p
```

ml_constant_init

`ml_constant_init` — batch 45 builtin.

```
my $result = ml_constant_init $x
# or in a pipeline:
@list |> map ml_constant_init |> p
```

ml_contrastive_loss

`ml_contrastive_loss` — batch 45 builtin.

```
my $result = ml_contrastive_loss $x
# or in a pipeline:
@list |> map ml_contrastive_loss |> p
```

ml_cosine_lr_schedule

`ml_cosine_lr_schedule` — batch 45 builtin.

```
my $result = ml_cosine_lr_schedule $x
# or in a pipeline:
@list |> map ml_cosine_lr_schedule |> p
```

ml_cross_entropy_loss

`ml_cross_entropy_loss` — batch 45 builtin.

```
my $result = ml_cross_entropy_loss $x
# or in a pipeline:
@list |> map ml_cross_entropy_loss |> p
```

ml_cutmix_box_iou

`ml_cutmix_box_iou` — batch 45 builtin.

```
my $result = ml_cutmix_box_iou $x
# or in a pipeline:
@list |> map ml_cutmix_box_iou |> p
```

ml_cyclic_lr_step

`ml_cyclic_lr_step` — batch 45 builtin.

```
my $result = ml_cyclic_lr_step $x
# or in a pipeline:
@list |> map ml_cyclic_lr_step |> p
```

ml_dice_loss_step

`ml_dice_loss_step` — batch 45 builtin.

```
my $result = ml_dice_loss_step $x
# or in a pipeline:
@list |> map ml_dice_loss_step |> p
```

ml_diou_loss_step

`ml_diou_loss_step` — batch 45 builtin.

```
my $result = ml_diou_loss_step $x
# or in a pipeline:
@list |> map ml_diou_loss_step |> p
```

ml_dropout_mask_prob

`ml_dropout_mask_prob` — batch 45 builtin.

```
my $result = ml_dropout_mask_prob $x
# or in a pipeline:
@list |> map ml_dropout_mask_prob |> p
```

ml_elu_step

`ml_elu_step` — batch 45 builtin.

```
my $result = ml_elu_step $x
# or in a pipeline:
@list |> map ml_elu_step |> p
```

ml_eos_logit_boost

`ml_eos_logit_boost` — batch 45 builtin.

```
my $result = ml_eos_logit_boost $x
# or in a pipeline:
@list |> map ml_eos_logit_boost |> p
```

ml_exponential_lr

`ml_exponential_lr` — batch 45 builtin.

```
my $result = ml_exponential_lr $x
# or in a pipeline:
@list |> map ml_exponential_lr |> p
```

ml_focal_loss_step

`ml_focal_loss_step` — batch 45 builtin.

```
my $result = ml_focal_loss_step $x
# or in a pipeline:
@list |> map ml_focal_loss_step |> p
```

ml_geglu_step

`ml_geglu_step` — batch 45 builtin.

```
my $result = ml_geglu_step $x
# or in a pipeline:
@list |> map ml_geglu_step |> p
```

ml_gelu_step

`ml_gelu_step` — batch 45 builtin.

```
my $result = ml_gelu_step $x
# or in a pipeline:
@list |> map ml_gelu_step |> p
```

ml_giou_loss_step

`ml_giou_loss_step` — batch 45 builtin.

```
my $result = ml_giou_loss_step $x
# or in a pipeline:
@list |> map ml_giou_loss_step |> p
```

ml_glorot_init_value

`ml_glorot_init_value` — batch 45 builtin.

```
my $result = ml_glorot_init_value $x
# or in a pipeline:
@list |> map ml_glorot_init_value |> p
```

ml_glu_step

`ml_glu_step` — batch 45 builtin.

```
my $result = ml_glu_step $x
# or in a pipeline:
@list |> map ml_glu_step |> p
```

ml_gradient_accumulate

`ml_gradient_accumulate` — batch 45 builtin.

```
my $result = ml_gradient_accumulate $x
# or in a pipeline:
@list |> map ml_gradient_accumulate |> p
```

ml_gradient_centralize

`ml_gradient_centralize` — batch 45 builtin.

```
my $result = ml_gradient_centralize $x
# or in a pipeline:
@list |> map ml_gradient_centralize |> p
```

ml_gradient_clip_norm

`ml_gradient_clip_norm` — batch 45 builtin.

```
my $result = ml_gradient_clip_norm $x
# or in a pipeline:
@list |> map ml_gradient_clip_norm |> p
```

ml_gradient_clip_value

`ml_gradient_clip_value` — batch 45 builtin.

```
my $result = ml_gradient_clip_value $x
# or in a pipeline:
@list |> map ml_gradient_clip_value |> p
```

ml_group_norm_step

`ml_group_norm_step` — batch 45 builtin.

```
my $result = ml_group_norm_step $x
# or in a pipeline:
@list |> map ml_group_norm_step |> p
```

ml_hard_sigmoid

ml_hard_sigmoid — batch 45 builtin.

```
my $result = ml_hard_sigmoid $x
# or in a pipeline:
@list |> map ml_hard_sigmoid |> p
```

ml_hard_tanh

ml_hard_tanh — batch 45 builtin.

```
my $result = ml_hard_tanh $x
# or in a pipeline:
@list |> map ml_hard_tanh |> p
```

ml_he_init_value

ml_he_init_value — batch 45 builtin.

```
my $result = ml_he_init_value $x
# or in a pipeline:
@list |> map ml_he_init_value |> p
```

ml_huber_loss_step

ml_huber_loss_step — batch 45 builtin.

```
my $result = ml_huber_loss_step $x
# or in a pipeline:
@list |> map ml_huber_loss_step |> p
```

ml_id_to_label_step

ml_id_to_label_step — batch 45 builtin.

```
my $result = ml_id_to_label_step $input
```

ml_instance_norm_step

ml_instance_norm_step — batch 45 builtin.

```
my $result = ml_instance_norm_step $x
# or in a pipeline:
@list |> map ml_instance_norm_step |> p
```

ml_inverse_sqrt_lr

ml_inverse_sqrt_lr — batch 45 builtin.

```
my $result = ml_inverse_sqrt_lr $x
# or in a pipeline:
@list |> map ml_inverse_sqrt_lr |> p
```

ml_iou_loss_step

ml_iou_loss_step — batch 45 builtin.

```
my $result = ml_iou_loss_step $x
# or in a pipeline:
@list |> map ml_iou_loss_step |> p
```

ml_kaiming_init

`ml_kaiming_init` — batch 45 builtin.

```
my $result = ml_kaiming_init $x
# or in a pipeline:
@list |> map ml_kaiming_init |> p
```

ml_kl_divergence_loss

`ml_kl_divergence_loss` — batch 45 builtin.

```
my $result = ml_kl_divergence_loss $x
# or in a pipeline:
@list |> map ml_kl_divergence_loss |> p
```

ml_l2_normalize_step

`ml_l2_normalize_step` — batch 45 builtin.

```
my $result = ml_l2_normalize_step $x
# or in a pipeline:
@list |> map ml_l2_normalize_step |> p
```

ml_label_smoothing

`ml_label_smoothing` — batch 45 builtin.

```
my $result = ml_label_smoothing $x
# or in a pipeline:
@list |> map ml_label_smoothing |> p
```

ml_label_to_id

`ml_label_to_id` — batch 45 builtin.

```
my $result = ml_label_to_id $input
```

ml_lamb_step

`ml_lamb_step` — batch 45 builtin.

```
my $result = ml_lamb_step $x
# or in a pipeline:
@list |> map ml_lamb_step |> p
```

ml_lars_step

`ml_lars_step` — batch 45 builtin.

```
my $result = ml_lars_step $x
# or in a pipeline:
@list |> map ml_lars_step |> p
```

ml_layer_norm_step

`ml_layer_norm_step` — batch 45 builtin.

```
my $result = ml_layer_norm_step $x
# or in a pipeline:
@list |> map ml_layer_norm_step |> p
```

ml_leaky_relu_step

`ml_leaky_relu_step` — batch 45 builtin.

```
my $result = ml_leaky_relu_step $x
# or in a pipeline:
@list |> map ml_leaky_relu_step |> p
```

ml_lecun_init_value

`ml_lecun_init_value` — batch 45 builtin.

```
my $result = ml_lecun_init_value $x
# or in a pipeline:
@list |> map ml_lecun_init_value |> p
```

ml_lion_step

`ml_lion_step` — batch 45 builtin.

```
my $result = ml_lion_step $x
# or in a pipeline:
@list |> map ml_lion_step |> p
```

ml_log_sigmoid

`ml_log_sigmoid` — batch 45 builtin.

```
my $result = ml_log_sigmoid $x
# or in a pipeline:
@list |> map ml_log_sigmoid |> p
```

ml_log_softmax_step

`ml_log_softmax_step` — batch 45 builtin.

```
my $result = ml_log_softmax_step $x
# or in a pipeline:
@list |> map ml_log_softmax_step |> p
```

ml_logsumexp_step

`ml_logsumexp_step` — batch 45 builtin.

```
my $result = ml_logsumexp_step $x
# or in a pipeline:
@list |> map ml_logsumexp_step |> p
```

ml_lookahead_step

`ml_lookahead_step` — batch 45 builtin.

```
my $result = ml_lookahead_step $x
# or in a pipeline:
@list |> map ml_lookahead_step |> p
```

ml_mish_step

`ml_mish_step` — batch 45 builtin.

```
my $result = ml_mish_step $x
# or in a pipeline:
@list |> map ml_mish_step |> p
```

ml_mixup_lambda

ml_mixup_lambda — batch 45 builtin.

```
my $result = ml_mixup_lambda $x
# or in a pipeline:
@list |> map ml_mixup_lambda |> p
```

ml_momentum_step

ml_momentum_step — batch 45 builtin.

```
my $result = ml_momentum_step $x
# or in a pipeline:
@list |> map ml_momentum_step |> p
```

ml_multihead_avg

ml_multihead_avg — batch 45 builtin.

```
my $result = ml_multihead_avg $x
# or in a pipeline:
@list |> map ml_multihead_avg |> p
```

ml_nadam_step

ml_nadam_step — batch 45 builtin.

```
my $result = ml_nadam_step $x
# or in a pipeline:
@list |> map ml_nadam_step |> p
```

ml_nesterov_momentum

ml_nesterov_momentum — batch 45 builtin.

```
my $result = ml_nesterov_momentum $x
# or in a pipeline:
@list |> map ml_nesterov_momentum |> p
```

ml_nucleus_sample_p

ml_nucleus_sample_p — batch 45 builtin.

```
my $result = ml_nucleus_sample_p $x
# or in a pipeline:
@list |> map ml_nucleus_sample_p |> p
```

ml_one_cycle_lr

ml_one_cycle_lr — batch 45 builtin.

```
my $result = ml_one_cycle_lr $x
# or in a pipeline:
@list |> map ml_one_cycle_lr |> p
```

ml_one_hot_index

ml_one_hot_index — batch 45 builtin.

```
my $result = ml_one_hot_index $x
# or in a pipeline:
@list |> map ml_one_hot_index |> p
```

ml_orthogonal_init

`ml_orthogonal_init` — batch 45 builtin.

```
my $result = ml_orthogonal_init $x
# or in a pipeline:
@list |> map ml_orthogonal_init |> p
```

ml_polynomial_lr

`ml_polynomial_lr` — batch 45 builtin.

```
my $result = ml_polynomial_lr $x
# or in a pipeline:
@list |> map ml_polynomial_lr |> p
```

ml_prelu_step

`ml_prelu_step` — batch 45 builtin.

```
my $result = ml_prelu_step $x
# or in a pipeline:
@list |> map ml_prelu_step |> p
```

ml_radam_step

`ml_radam_step` — batch 45 builtin.

```
my $result = ml_radam_step $x
# or in a pipeline:
@list |> map ml_radam_step |> p
```

ml_random_erasing_step

`ml_random_erasing_step` — batch 45 builtin.

```
my $result = ml_random_erasing_step $x
# or in a pipeline:
@list |> map ml_random_erasing_step |> p
```

ml_relu_step

`ml_relu_step` — batch 45 builtin.

```
my $result = ml_relu_step $x
# or in a pipeline:
@list |> map ml_relu_step |> p
```

ml_repetition_penalty

`ml_repetition_penalty` — batch 45 builtin.

```
my $result = ml_repetition_penalty $x
# or in a pipeline:
@list |> map ml_repetition_penalty |> p
```

ml_rms_norm_step

`ml_rms_norm_step` — batch 45 builtin.

```
my $result = ml_rms_norm_step $x
# or in a pipeline:
@list |> map ml_rms_norm_step |> p
```

ml_rmsprop_step

`ml_rmsprop_step` — batch 45 builtin.

```
my $result = ml_rmsprop_step $x
# or in a pipeline:
@list |> map ml_rmsprop_step |> p
```

ml_scaled_dot_product

`ml_scaled_dot_product` — batch 45 builtin.

```
my $result = ml_scaled_dot_product $x
# or in a pipeline:
@list |> map ml_scaled_dot_product |> p
```

ml_selu_step

`ml_selu_step` — batch 45 builtin.

```
my $result = ml_selu_step $x
# or in a pipeline:
@list |> map ml_selu_step |> p
```

ml_sgd_step

`ml_sgd_step` — batch 45 builtin.

```
my $result = ml_sgd_step $x
# or in a pipeline:
@list |> map ml_sgd_step |> p
```

ml_shampoo_step

`ml_shampoo_step` — batch 45 builtin.

```
my $result = ml_shampoo_step $x
# or in a pipeline:
@list |> map ml_shampoo_step |> p
```

ml_silu_step

`ml_silu_step` — batch 45 builtin.

```
my $result = ml_silu_step $x
# or in a pipeline:
@list |> map ml_silu_step |> p
```

ml_smooth_l1_loss

`ml_smooth_l1_loss` — batch 45 builtin.

```
my $result = ml_smooth_l1_loss $x
# or in a pipeline:
@list |> map ml_smooth_l1_loss |> p
```

ml_softmax_temperature

`ml_softmax_temperature` — batch 45 builtin.

```
my $result = ml_softmax_temperature $x
# or in a pipeline:
@list |> map ml_softmax_temperature |> p
```

ml_softplus_step

`ml_softplus_step` — batch 45 builtin.

```
my $result = ml_softplus_step $x
# or in a pipeline:
@list |> map ml_softplus_step |> p
```

ml_softsign_step

`ml_softsign_step` — batch 45 builtin.

```
my $result = ml_softsign_step $x
# or in a pipeline:
@list |> map ml_softsign_step |> p
```

ml_sophia_step

`ml_sophia_step` — batch 45 builtin.

```
my $result = ml_sophia_step $x
# or in a pipeline:
@list |> map ml_sophia_step |> p
```

ml_spectral_norm_step

`ml_spectral_norm_step` — batch 45 builtin.

```
my $result = ml_spectral_norm_step $x
# or in a pipeline:
@list |> map ml_spectral_norm_step |> p
```

ml_step_lr_schedule

`ml_step_lr_schedule` — batch 45 builtin.

```
my $result = ml_step_lr_schedule $x
# or in a pipeline:
@list |> map ml_step_lr_schedule |> p
```

ml_swiglu_step

`ml_swiglu_step` — batch 45 builtin.

```
my $result = ml_swiglu_step $x
# or in a pipeline:
@list |> map ml_swiglu_step |> p
```

ml_swish_step

`ml_swish_step` — batch 45 builtin.

```
my $result = ml_swish_step $x
# or in a pipeline:
@list |> map ml_swish_step |> p
```

ml_temperature_decay

`ml_temperature_decay` — batch 45 builtin.

```
my $result = ml_temperature_decay $x
# or in a pipeline:
@list |> map ml_temperature_decay |> p
```

ml_to_cups

`ml_to_cups` — file stat / path builtin.

```
my $result = ml_to_cups $input
```

ml_to_liters

`ml_to_liters` — file stat / path builtin.

```
my $result = ml_to_liters $input
```

ml_token_logit_top_k

`ml_token_logit_top_k` — batch 45 builtin.

```
my $result = ml_token_logit_top_k $x
# or in a pipeline:
@list |> map ml_token_logit_top_k |> p
```

ml_topk_argmax

`ml_topk_argmax` — batch 45 builtin.

```
my $result = ml_topk_argmax $x
# or in a pipeline:
@list |> map ml_topk_argmax |> p
```

ml_triplet_loss_step

`ml_triplet_loss_step` — batch 45 builtin.

```
my $result = ml_triplet_loss_step $x
# or in a pipeline:
@list |> map ml_triplet_loss_step |> p
```

ml_truncnormal_init

`ml_truncnormal_init` — batch 45 builtin.

```
my $result = ml_truncnormal_init $x
# or in a pipeline:
@list |> map ml_truncnormal_init |> p
```

ml_uniform_init

`ml_uniform_init` — batch 45 builtin.

```
my $result = ml_uniform_init $x
# or in a pipeline:
@list |> map ml_uniform_init |> p
```

ml_warmup_lr_step

`ml_warmup_lr_step` — batch 45 builtin.

```
my $result = ml_warmup_lr_step $x
# or in a pipeline:
@list |> map ml_warmup_lr_step |> p
```

ml_weight_decay_step

ml_weight_decay_step — batch 45 builtin.

```
my $result = ml_weight_decay_step $x
# or in a pipeline:
@list |> map ml_weight_decay_step |> p
```

ml_weight_norm_step

ml_weight_norm_step — batch 45 builtin.

```
my $result = ml_weight_norm_step $x
# or in a pipeline:
@list |> map ml_weight_norm_step |> p
```

ml_xavier_init_value

ml_xavier_init_value — batch 45 builtin.

```
my $result = ml_xavier_init_value $x
# or in a pipeline:
@list |> map ml_xavier_init_value |> p
```

ml_yogi_step

ml_yogi_step — batch 45 builtin.

```
my $result = ml_yogi_step $x
# or in a pipeline:
@list |> map ml_yogi_step |> p
```

ml_zero_init

ml_zero_init — batch 45 builtin.

```
my $result = ml_zero_init $x
# or in a pipeline:
@list |> map ml_zero_init |> p
```

mle_exponential_log_lik

mle_exponential_log_lik — batch 36 builtin.

```
my $result = mle_exponential_log_lik $x
# or in a pipeline:
@list |> map mle_exponential_log_lik |> p
```

mle_normal_log_lik

mle_normal_log_lik — batch 36 builtin.

```
my $result = mle_normal_log_lik $x
# or in a pipeline:
@list |> map mle_normal_log_lik |> p
```

mle_poisson_log_lik

mle_poisson_log_lik — batch 36 builtin.

```
my $result = mle_poisson_log_lik $x
# or in a pipeline:
@list |> map mle_poisson_log_lik |> p
```

mm

`mm` — extended stdlib builtin. Alias for `matrix_mul`.

```
my $result = mm $x
# or in a pipeline:
@list |> map mm |> p
```

mmap

`mmap` — matrix operations (uncategorized batch) builtin. Alias for `matrix_map`.

```
my $result = mmap $x
# or in a pipeline:
@list |> map mmap |> p
```

mmax

`mmax` — matrix operations (uncategorized batch) builtin. Alias for `matrix_max`.

```
my $result = mmax $x
# or in a pipeline:
@list |> map mmax |> p
```

mmhg_to_pa

`mmhg_to_pa` — batch 30 builtin.

```
my $result = mmhg_to_pa $input
```

mmin

`mmin` — matrix operations (uncategorized batch) builtin. Alias for `matrix_min`.

```
my $result = mmin $x
# or in a pipeline:
@list |> map mmin |> p
```

mmult

`mmult` — extended stdlib builtin. Alias for `matrix_mult`.

```
my $result = mmult $x
# or in a pipeline:
@list |> map mmult |> p
```

mmx

`mmx` — python/ruby stdlib builtin. Alias for `minmax`.

```
my $result = mmx $x
# or in a pipeline:
@list |> map mmx |> p
```

mmxb

`mmxb` — python/ruby stdlib builtin. Alias for `minmax_by`.

```
my $result = mmxb $x
# or in a pipeline:
@list |> map mmxb |> p
```

mob

`mob` — extended stdlib batch 3 builtin. Alias for `mobius`.

```
my $result = mob $x
# or in a pipeline:
@list |> map mob |> p
```

mobius

`mobius` — extended stdlib batch 3 builtin.

```
my $result = mobius $x
# or in a pipeline:
@list |> map mobius |> p
```

mobius_function_pair

`mobius_function_pair` — batch 37 builtin.

```
my $result = mobius_function_pair $x
# or in a pipeline:
@list |> map mobius_function_pair |> p
```

mobius_inversion_step

`mobius_inversion_step` — batch 37 builtin.

```
my $result = mobius_inversion_step $x
# or in a pipeline:
@list |> map mobius_inversion_step |> p
```

mobius_poset_two

`mobius_poset_two` — batch 37 builtin.

```
my $result = mobius_poset_two $x
# or in a pipeline:
@list |> map mobius_poset_two |> p
```

mod_exp

`mod_exp` (aliases `modexp`, `powmod`) computes modular exponentiation: base^exp mod m, using fast binary exponentiation.

```
p powmod(2, 10, 1000)  # 24 (2^10 mod 1000)
p powmod(3, 13, 50)    # 7
```

mod_inv

`mod_inv` (alias `modinv`) computes the modular multiplicative inverse via extended Euclidean algorithm. Errors if no inverse exists.

```
p modinv(3, 7)   # 5 (3*5 □ 1 mod 7)
```

mod_op

`mod_op` — trig / math (batch 2) builtin.

```
my $result = mod_op $x
# or in a pipeline:
@list |> map mod_op |> p
```

mod_sph_bessel_i0

`mod_sph_bessel_i0` — batch 29 builtin.

```
my $result = mod_sph_bessel_i0 $x
# or in a pipeline:
@list |> map mod_sph_bessel_i0 |> p
```

mod_sph_bessel_i1

`mod_sph_bessel_i1` — batch 29 builtin.

```
my $result = mod_sph_bessel_i1 $x
# or in a pipeline:
@list |> map mod_sph_bessel_i1 |> p
```

mod_sph_bessel_k0

`mod_sph_bessel_k0` — batch 29 builtin.

```
my $result = mod_sph_bessel_k0 $x
# or in a pipeline:
@list |> map mod_sph_bessel_k0 |> p
```

moddur

`moddur` — finance (extended) builtin. Alias for `modified_duration`.

```
my $result = moddur $x
# or in a pipeline:
@list |> map moddur |> p
```

mode_agg

`mode_agg` — batch 70 builtin.

```
my $result = mode_agg $x
# or in a pipeline:
@list |> map mode_agg |> p
```

mode_list

`mode_list` — list helpers (batch 4) builtin.

```
my $result = mode_list $x
# or in a pipeline:
@list |> map mode_list |> p
```

mode_pitches_dorian

`mode_pitches_dorian` — batch 64 builtin.

```
my $result = mode_pitches_dorian $x
# or in a pipeline:
@list |> map mode_pitches_dorian |> p
```

mode_pitches_lydian

`mode_pitches_lydian` — batch 64 builtin.

```
my $result = mode_pitches_lydian $x
# or in a pipeline:
@list |> map mode_pitches_lydian |> p
```

mode_pitches_phrygian

`mode_pitches_phrygian` — batch 64 builtin.

```
my $result = mode_pitches_phrygian $x
# or in a pipeline:
@list |> map mode_pitches_phrygian |> p
```

mode_stat

`mode_stat` — extended stdlib builtin.

```
my $result = mode_stat $x
# or in a pipeline:
@list |> map mode_stat |> p
```

mode_val

`mode_val` — trivial numeric / predicate builtins builtin.

```
my $result = mode_val $x
# or in a pipeline:
@list |> map mode_val |> p
```

model_checking_ctl

`model_checking_ctl` — batch 67 builtin.

```
my $result = model_checking_ctl $x
# or in a pipeline:
@list |> map model_checking_ctl |> p
```

model_checking_ltl

`model_checking_ltl` — batch 67 builtin.

```
my $result = model_checking_ltl $x
# or in a pipeline:
@list |> map model_checking_ltl |> p
```

modhex_encode

`modhex_encode` — batch 62 builtin.

```
my $result = modhex_encode $x
# or in a pipeline:
@list |> map modhex_encode |> p
```

modified_duration

`modified_duration($face, $coupon_rate, $ytm, $periods)` (alias `mod_dur`) — computes modified duration, which directly measures price sensitivity to yield changes. Equal to Macaulay duration / (1 + ytm).

```
p mod_dur(1000, 0.05, 0.05, 10)   # ~7.3
```

modified_irr

`modified_irr` — batch 10 builtin.

```
my $result = modified_irr $x
# or in a pipeline:
@list |> map modified_irr |> p
```

modified_midpoint_ode

`modified_midpoint_ode` — batch 19 builtin.

```
my $result = modified_midpoint_ode $x
# or in a pipeline:
@list |> map modified_midpoint_ode |> p
```

modified_sharpe

`modified_sharpe` — batch 20 builtin.

```
my $result = modified_sharpe $x
# or in a pipeline:
@list |> map modified_sharpe |> p
```

modmidpoint

`modmidpoint` — batch 19 builtin. Alias for `modified_midpoint_ode`.

```
my $result = modmidpoint $x
# or in a pipeline:
@list |> map modmidpoint |> p
```

modular_data_s_value

`modular_data_s_value` — batch 37 builtin.

```
my $result = modular_data_s_value $x
# or in a pipeline:
@list |> map modular_data_s_value |> p
```

modular_data_t_value

`modular_data_t_value` — batch 37 builtin.

```
my $result = modular_data_t_value $x
# or in a pipeline:
@list |> map modular_data_t_value |> p
```

modularity_q

`modularity_q` — batches 14-16 builtin.

```
my $result = modularity_q $x
# or in a pipeline:
@list |> map modularity_q |> p
```

modularity_score

`modularity_score` — batch 75 builtin.

```
my $result = modularity_score $x
# or in a pipeline:
@list |> map modularity_score |> p
```

moduli_dimension_curves

`moduli_dimension_curves` — batch 37 builtin.

```
my $result = moduli_dimension_curves $x
# or in a pipeline:
@list |> map moduli_dimension_curves |> p
```

molality

`molality` — batch 12 builtin.

```
my $result = molality $x
# or in a pipeline:
@list |> map molality |> p
```

molarity

`molarity` — batch 12 builtin.

```
my $result = molarity $x
# or in a pipeline:
@list |> map molarity |> p
```

molarity_dilution

`molarity_dilution` — batch 63 builtin.

```
my $result = molarity_dilution $x
# or in a pipeline:
@list |> map molarity_dilution |> p
```

mole_fraction

`mole_fraction` — batch 12 builtin.

```
my $result = mole_fraction $x
# or in a pipeline:
@list |> map mole_fraction |> p
```

molecular_weight_compound

`molecular_weight_compound` — batch 63 builtin.

```
my $result = molecular_weight_compound $x
# or in a pipeline:
@list |> map molecular_weight_compound |> p
```

moles_from_mass

`moles_from_mass` — batch 21 builtin.

```
my $result = moles_from_mass $x
# or in a pipeline:
@list |> map moles_from_mass |> p
```

mollweide_project

`mollweide_project` — batches 14-16 builtin.

```
my $result = mollweide_project $x
# or in a pipeline:
@list |> map mollweide_project |> p
```

moment_magnitude_mw

`moment_magnitude_mw` — batch 65 builtin.

```
my $result = moment_magnitude_mw $x
# or in a pipeline:
@list |> map moment_magnitude_mw |> p
```

momentum

`momentum($mass, $velocity)` — compute momentum p = mv (kg⋅m/s).

```
p momentum(10, 5)   # 50 kg·m/s
```

mones

`mones` — extended stdlib builtin. Alias for `matrix_ones`.

```
my $result = mones $x
# or in a pipeline:
@list |> map mones |> p
```

monin_obukhov_length

`monin_obukhov_length` — batch 42 builtin.

```
my $result = monin_obukhov_length $x
# or in a pipeline:
@list |> map monin_obukhov_length |> p
```

monobit_test

`monobit_test` — batch 26 builtin.

```
my $result = monobit_test $x
# or in a pipeline:
@list |> map monobit_test |> p
```

monopoly_lerner

`monopoly_lerner` — batch 73 builtin.

```
my $result = monopoly_lerner $x
# or in a pipeline:
@list |> map monopoly_lerner |> p
```

montecarlo_returns_step

`montecarlo_returns_step` — batch 40 builtin.

```
my $result = montecarlo_returns_step $x
# or in a pipeline:
@list |> map montecarlo_returns_step |> p
```

month_name

`month_name` — date helpers builtin.

```
my $result = month_name $x
# or in a pipeline:
@list |> map month_name |> p
```

month_short

`month_short` — date helpers builtin.

```
my $result = month_short $x
# or in a pipeline:
@list |> map month_short |> p
```

months_between

`months_between` — batch 9 builtin.

```
my $result = months_between $x
# or in a pipeline:
@list |> map months_between |> p
```

montreal_protocol_track

`montreal_protocol_track` — batch 42 builtin.

```
my $result = montreal_protocol_track $x
# or in a pipeline:
@list |> map montreal_protocol_track |> p
```

moon_phase

`moon_phase` — batch 9 builtin.

```
my $result = moon_phase $x
# or in a pipeline:
@list |> map moon_phase |> p
```

moon_phase_age

`moon_phase_age` — batch 55 builtin.

```
my $result = moon_phase_age $x
# or in a pipeline:
@list |> map moon_phase_age |> p
```

moon_position_low

`moon_position_low` — batch 55 builtin.

```
my $result = moon_position_low $x
# or in a pipeline:
@list |> map moon_position_low |> p
```

morse

`morse` — encoding / phonetics builtin. Alias for `morse_encode`.

```
my $result = morse $x
# or in a pipeline:
@list |> map morse |> p
```

morse_decode

`morse_decode` — encoding / phonetics builtin.

```
my $result = morse_decode $x
# or in a pipeline:
@list |> map morse_decode |> p
```

morse_encode

`morse_encode` — encoding / phonetics builtin.

```
my $result = morse_encode $x
# or in a pipeline:
@list |> map morse_encode |> p
```

mortality_table_q

`mortality_table_q` — batch 60 builtin.

```
my $result = mortality_table_q $x
# or in a pipeline:
@list |> map mortality_table_q |> p
```

mortgage_payment

`mortgage_payment` — physics formulas builtin.

```
my $result = mortgage_payment $x
# or in a pipeline:
@list |> map mortgage_payment |> p
```

most_common

`most_common` — list helpers (batch 4) builtin.

```
my $result = most_common $x
# or in a pipeline:
@list |> map most_common |> p
```

motion_blur_kernel

`motion_blur_kernel` — batches 14-16 builtin.

```
my $result = motion_blur_kernel $x
# or in a pipeline:
@list |> map motion_blur_kernel |> p
```

motor_torque

`motor_torque` — batch 12 builtin.

```
my $result = motor_torque $x
# or in a pipeline:
@list |> map motor_torque |> p
```

mott_schottky_capacitance

`mott_schottky_capacitance` — batch 38 builtin.

```
my $result = mott_schottky_capacitance $x
# or in a pipeline:
@list |> map mott_schottky_capacitance |> p
```

movavg

`movavg` — math / numeric (uncategorized batch) builtin. Alias for `moving_average`.

```
my $result = movavg $x
# or in a pipeline:
@list |> map movavg |> p
```

moving_average

`moving_average` — math / numeric (uncategorized batch) builtin.

```
my $result = moving_average $x
# or in a pipeline:
@list |> map moving_average |> p
```

moving_average_filter

`moving_average_filter` — batch 78 builtin.

```
my $result = moving_average_filter $x
# or in a pipeline:
@list |> map moving_average_filter |> p
```

mpeg_quant_value

`mpeg_quant_value` — batch 40 builtin.

```
my $result = mpeg_quant_value $x
# or in a pipeline:
@list |> map mpeg_quant_value |> p
```

mph_to_kmh

`mph_to_kmh` — batch 30 builtin.

```
my $result = mph_to_kmh $input
```

mplanck

`mplanck` — physics constants builtin. Alias for `planck_mass`.

```
my $result = mplanck $x
# or in a pipeline:
@list |> map mplanck |> p
```

mpow

`mpow` — matrix operations (uncategorized batch) builtin. Alias for `matrix_power`.

```
my $result = mpow $x
# or in a pipeline:
@list |> map mpow |> p
```

mps_to_kmh

`mps_to_kmh` — batch 30 builtin.

```
my $result = mps_to_kmh $input
```

mrow

`mrow` — extended stdlib builtin. Alias for `matrix_row`.

```
my $result = mrow $x
# or in a pipeline:
@list |> map mrow |> p
```

ms_to_ns

`ms_to_ns` — file stat / path builtin.

```
my $result = ms_to_ns $input
```

ms_to_s

`ms_to_s` — file stat / path builtin.

```
my $result = ms_to_s $input
```

msa_column_entropy

`msa_column_entropy` — batch 33 builtin.

```
my $result = msa_column_entropy $x
# or in a pipeline:
@list |> map msa_column_entropy |> p
```

mscal

`mscal` — extended stdlib builtin. Alias for `matrix_scalar`.

```
my $result = mscal $x
# or in a pipeline:
@list |> map mscal |> p
```

mse

`mse` — math functions builtin.

```
my $result = mse $x
# or in a pipeline:
@list |> map mse |> p
```

mset

`mset` — batch 71 builtin.

```
my $result = mset $x
# or in a pipeline:
@list |> map mset |> p
```

msgctl

`msgctl` — sysv ipc builtin.

```
my $result = msgctl $x
# or in a pipeline:
@list |> map msgctl |> p
```

msgget

`msgget` — sysv ipc builtin.

```
my $result = msgget $x
# or in a pipeline:
@list |> map msgget |> p
```

msgpack_pack_int

`msgpack_pack_int` — batch 62 builtin.

```
my $result = msgpack_pack_int $x
# or in a pipeline:
@list |> map msgpack_pack_int |> p
```

msgpack_pack_str

`msgpack_pack_str` — batch 62 builtin.

```
my $result = msgpack_pack_str $x
# or in a pipeline:
@list |> map msgpack_pack_str |> p
```

msgrcv

`msgrcv` — sysv ipc builtin.

```
my $result = msgrcv $x
# or in a pipeline:
@list |> map msgrcv |> p
```

msgsnd

`msgsnd` — sysv ipc builtin.

```
my $result = msgsnd $x
# or in a pipeline:
@list |> map msgsnd |> p
```

mshape

`mshape` — extended stdlib builtin. Alias for `matrix_shape`.

```
my $result = mshape $x
# or in a pipeline:
@list |> map mshape |> p
```

msorted

`msorted` — matrix operations (uncategorized batch) builtin. Alias for `merge_sorted`.

```
my $result = msorted $x
# or in a pipeline:
@list |> map msorted |> p
```

mst

`prim_mst` (aliases `mst`, `prim`) computes the minimum spanning tree weight via Prim's algorithm. Takes an adjacency matrix.

```
p mst([[0,2,0],[2,0,3],[0,3,0]])  # 5
```

mstat

`mstat` — extended stdlib builtin. Alias for `mode_stat`.

```
my $result = mstat $x
# or in a pipeline:
@list |> map mstat |> p
```

msub

`msub` — extended stdlib builtin. Alias for `matrix_sub`.

```
my $result = msub $x
# or in a pipeline:
@list |> map msub |> p
```

msum

`msum` — matrix operations (uncategorized batch) builtin. Alias for `matrix_sum`.

```
my $result = msum $x
# or in a pipeline:
@list |> map msum |> p
```

msun

`sun_mass` (alias `msun`) — M☉ 1.989×10³⁰ kg. Mass of Sun (solar mass).

```
p msun   # 1.98892e30 kg
```

mt19937_temper

`mt19937_temper` — batch 26 builtin.

```
my $result = mt19937_temper $x
# or in a pipeline:
@list |> map mt19937_temper |> p
```

mtf_decode

`mtf_decode` — batches 14-16 builtin.

```
my $result = mtf_decode $x
# or in a pipeline:
@list |> map mtf_decode |> p
```

mtf_encode

`mtf_encode` — batches 14-16 builtin.

```
my $result = mtf_encode $x
# or in a pipeline:
@list |> map mtf_encode |> p
```

mtime

`mtime` — file stat / path builtin. Alias for `file_mtime`.

```
my $result = mtime $x
# or in a pipeline:
@list |> map mtime |> p
```

mtrace

`mtrace` — extended stdlib builtin. Alias for `matrix_trace`.

```
my $result = mtrace $x
# or in a pipeline:
@list |> map mtrace |> p
```

mtrans

`mtrans` — matrix operations (uncategorized batch) builtin. Alias for `matrix_transpose`.

```
my $result = mtrans $x
# or in a pipeline:
@list |> map mtrans |> p
```

mu0

`mu0` — physics constants builtin. Alias for `vacuum_permeability`.

```
my $result = mu0 $x
# or in a pipeline:
@list |> map mu0 |> p
```

mu_law_decode

`mu_law_decode` — batch 25 builtin.

```
my $result = mu_law_decode $x
# or in a pipeline:
@list |> map mu_law_decode |> p
```

mu_law_encode

`mu_law_encode` — batch 25 builtin.

```
my $result = mu_law_encode $x
# or in a pipeline:
@list |> map mu_law_encode |> p
```

mub

`mub` — physics constants builtin. Alias for `bohr_magneton`.

```
my $result = mub $x
# or in a pipeline:
@list |> map mub |> p
```

muller

`muller` — batch 19 builtin. Alias for `muller_root`.

```
my $result = muller $x
# or in a pipeline:
@list |> map muller |> p
```

muller_root

`muller_root` — batch 19 builtin.

```
my $result = muller_root $x
# or in a pipeline:
@list |> map muller_root |> p
```

multi_decrement_q

`multi_decrement_q` — batch 60 builtin.

```
my $result = multi_decrement_q $x
# or in a pipeline:
@list |> map multi_decrement_q |> p
```

multi_knapsack_step

`multi_knapsack_step` — batch 41 builtin.

```
my $result = multi_knapsack_step $x
# or in a pipeline:
@list |> map multi_knapsack_step |> p
```

multi_state_pij

`multi_state_pij` — batch 60 builtin.

```
my $result = multi_state_pij $x
# or in a pipeline:
@list |> map multi_state_pij |> p
```

multidim_packing_step

`multidim_packing_step` — batch 41 builtin.

```
my $result = multidim_packing_step $x
# or in a pipeline:
@list |> map multidim_packing_step |> p
```

multinom

`multinom` — math / numeric (uncategorized batch) builtin. Alias for `multinomial`.

```
my $result = multinom $x
# or in a pipeline:
@list |> map multinom |> p
```

multinomial

`multinomial` — math / numeric (uncategorized batch) builtin.

```
my $result = multinomial $x
# or in a pipeline:
@list |> map multinomial |> p
```

multinomial_count

`multinomial_count` — batch 9 builtin. Alias for `multiset_permutations_count`.

```
my $result = multinomial_count $x
# or in a pipeline:
@list |> map multinomial_count |> p
```

multinomial_logit_prob

multinomial_logit_prob — batch 36 builtin.

```
my $result = multinomial_logit_prob $x
# or in a pipeline:
@list |> map multinomial_logit_prob |> p
```

multiset_difference

multiset_difference — batch 10 builtin.

```
my $result = multiset_difference $x
# or in a pipeline:
@list |> map multiset_difference |> p
```

multiset_intersection

multiset_intersection — batch 10 builtin.

```
my $result = multiset_intersection $x
# or in a pipeline:
@list |> map multiset_intersection |> p
```

multiset_permutations_count

multiset_permutations_count — batch 9 builtin.

```
my $result = multiset_permutations_count $x
# or in a pipeline:
@list |> map multiset_permutations_count |> p
```

multiset_union

multiset_union — batch 10 builtin.

```
my $result = multiset_union $x
# or in a pipeline:
@list |> map multiset_union |> p
```

mun

mun — physics constants builtin. Alias for nuclear_magneton.

```
my $result = mun $x
# or in a pipeline:
@list |> map mun |> p
```

murmurhash3_x32

murmurhash3_x32 — batch 11 builtin.

```
my $result = murmurhash3_x32 $x
# or in a pipeline:
@list |> map murmurhash3_x32 |> p
```

mutation_bit_flip_prob

mutation_bit_flip_prob — batch 41 builtin.

```
my $result = mutation_bit_flip_prob $x
# or in a pipeline:
@list |> map mutation_bit_flip_prob |> p
```

mutation_rate

`mutation_rate` — batch 33 builtin.

```
my $result = mutation_rate $x
# or in a pipeline:
@list |> map mutation_rate |> p
```

mutual_information_step

`mutual_information_step` — batch 40 builtin.

```
my $result = mutual_information_step $x
# or in a pipeline:
@list |> map mutual_information_step |> p
```

myerson_optimal

`myerson_optimal` — batch 73 builtin.

```
my $result = myerson_optimal $x
# or in a pipeline:
@list |> map myerson_optimal |> p
```

mzeros

`mzeros` — extended stdlib builtin. Alias for `matrix_zeros`.

```
my $result = mzeros $x
# or in a pipeline:
@list |> map mzeros |> p
```

n_ball_volume

`n_ball_volume` — batch 28 builtin.

```
my $result = n_ball_volume $x
# or in a pipeline:
@list |> map n_ball_volume |> p
```

nanoid

`nanoid` — id helpers builtin.

```
my $result = nanoid $x
# or in a pipeline:
@list |> map nanoid |> p
```

nao_index_step

`nao_index_step` — batch 42 builtin.

```
my $result = nao_index_step $x
# or in a pipeline:
@list |> map nao_index_step |> p
```

nash_bargaining_solution

nash_bargaining_solution — batch 43 builtin.

```
my $result = nash_bargaining_solution $x
# or in a pipeline:
@list |> map nash_bargaining_solution |> p
```

nash_equilibrium_pair

nash_equilibrium_pair — batch 43 builtin.

```
my $result = nash_equilibrium_pair $x
# or in a pipeline:
@list |> map nash_equilibrium_pair |> p
```

nash_social_welfare

nash_social_welfare — batch 43 builtin.

```
my $result = nash_social_welfare $x
# or in a pipeline:
@list |> map nash_social_welfare |> p
```

nato

nato — encoding / phonetics builtin. Alias for nato_phonetic.

```
my $result = nato $x
# or in a pipeline:
@list |> map nato |> p
```

nato_phonetic

nato_phonetic — encoding / phonetics builtin.

```
my $result = nato_phonetic $input
```

nb_bernoulli_likelihood

nb_bernoulli_likelihood — batch 27 builtin.

```
my $result = nb_bernoulli_likelihood $x
# or in a pipeline:
@list |> map nb_bernoulli_likelihood |> p
```

nb_gaussian_likelihood

nb_gaussian_likelihood — batch 27 builtin.

```
my $result = nb_gaussian_likelihood $x
# or in a pipeline:
@list |> map nb_gaussian_likelihood |> p
```

nb_multinomial_log_likelihood

nb_multinomial_log_likelihood — batch 27 builtin.

```
my $result = nb_multinomial_log_likelihood $x
# or in a pipeline:
@list |> map nb_multinomial_log_likelihood |> p
```

nbe_normalize

`nbe_normalize` — batch 67 builtin.

```
my $result = nbe_normalize $x
# or in a pipeline:
@list |> map nbe_normalize |> p
```

ncol

`ncol` — number of columns in a matrix.

```
p ncol([[1,2,3],[4,5,6]])  # 3
```

ndivs

`ndivs` — math / numeric (uncategorized batch) builtin. Alias for `num_divisors`.

```
my $result = ndivs $x
# or in a pipeline:
@list |> map ndivs |> p
```

ne_from_variance

`ne_from_variance` — batch 33 builtin.

```
my $result = ne_from_variance $x
# or in a pipeline:
@list |> map ne_from_variance |> p
```

nearest_insertion_step

`nearest_insertion_step` — batch 41 builtin.

```
my $result = nearest_insertion_step $x
# or in a pipeline:
@list |> map nearest_insertion_step |> p
```

nearest_neighbor_tour_step

`nearest_neighbor_tour_step` — batch 41 builtin.

```
my $result = nearest_neighbor_tour_step $x
# or in a pipeline:
@list |> map nearest_neighbor_tour_step |> p
```

necklace_count

`necklace_count` — batch 9 builtin.

```
my $result = necklace_count $x
# or in a pipeline:
@list |> map necklace_count |> p
```

needleman_wunsch_score

`needleman_wunsch_score` — batch 33 builtin.

```
my $result = needleman_wunsch_score $x
# or in a pipeline:
@list |> map needleman_wunsch_score |> p
```

neg_inf

`neg_inf` — math / numeric (uncategorized batch) builtin.

```
my $result = neg_inf $x
# or in a pipeline:
@list |> map neg_inf |> p
```

neg_n

`neg_n` — trivial numeric / predicate builtins builtin. Alias for `negative`.

```
my $result = neg_n $x
# or in a pipeline:
@list |> map neg_n |> p
```

negate

`negate` — trivial numeric / predicate builtins builtin.

```
my $result = negate $x
# or in a pipeline:
@list |> map negate |> p
```

negate_each

`negate_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = negate_each $x
# or in a pipeline:
@list |> map negate_each |> p
```

negative

`negative` — trivial numeric / predicate builtins builtin.

```
my $result = negative $x
# or in a pipeline:
@list |> map negative |> p
```

nei_genetic_distance

`nei_genetic_distance` — batch 22 builtin.

```
my $result = nei_genetic_distance $x
# or in a pipeline:
@list |> map nei_genetic_distance |> p
```

neither

`neither` — boolean combinators builtin.

```
my $result = neither $x
# or in a pipeline:
@list |> map neither |> p
```

nelder_mead_contract

`nelder_mead_contract` — batch 19 builtin.

```
my $result = nelder_mead_contract $x
# or in a pipeline:
@list |> map nelder_mead_contract |> p
```

nelder_mead_expand

`nelder_mead_expand` — batch 19 builtin.

```
my $result = nelder_mead_expand $x
# or in a pipeline:
@list |> map nelder_mead_expand |> p
```

nelder_mead_reflect

`nelder_mead_reflect` — batch 19 builtin.

```
my $result = nelder_mead_reflect $x
# or in a pipeline:
@list |> map nelder_mead_reflect |> p
```

nemhauser_wolsey_bound

`nemhauser_wolsey_bound` — batch 41 builtin.

```
my $result = nemhauser_wolsey_bound $x
# or in a pipeline:
@list |> map nemhauser_wolsey_bound |> p
```

ner_bilou_decode

`ner_bilou_decode` — batch 69 builtin.

```
my $result = ner_bilou_decode $x
# or in a pipeline:
@list |> map ner_bilou_decode |> p
```

nernst_diffusion_layer

`nernst_diffusion_layer` — batch 38 builtin.

```
my $result = nernst_diffusion_layer $x
# or in a pipeline:
@list |> map nernst_diffusion_layer |> p
```

nernst_einstein_diffusivity

`nernst_einstein_diffusivity` — batch 38 builtin.

```
my $result = nernst_einstein_diffusivity $x
# or in a pipeline:
@list |> map nernst_einstein_diffusivity |> p
```

nernst_planck_flux

`nernst_planck_flux` — batch 38 builtin.

```
my $result = nernst_planck_flux $x
# or in a pipeline:
@list |> map nernst_planck_flux |> p
```

nernst_potential_full

`nernst_potential_full` — batch 38 builtin.

```
my $result = nernst_potential_full $x
# or in a pipeline:
@list |> map nernst_potential_full |> p
```

nerve_complex_count

`nerve_complex_count` — batch 37 builtin.

```
my $result = nerve_complex_count $x
# or in a pipeline:
@list |> map nerve_complex_count |> p
```

nesterov_accelerate_step

`nesterov_accelerate_step` — batch 41 builtin.

```
my $result = nesterov_accelerate_step $x
# or in a pipeline:
@list |> map nesterov_accelerate_step |> p
```

nesterov_step

`nesterov_step` — batch 19 builtin.

```
my $result = nesterov_step $x
# or in a pipeline:
@list |> map nesterov_step |> p
```

net_aiad_step

`net_aiad_step` — batch 49 builtin.

```
my $result = net_aiad_step $x
# or in a pipeline:
@list |> map net_aiad_step |> p
```

net_aimd_step

`net_aimd_step` — batch 49 builtin.

```
my $result = net_aimd_step $x
# or in a pipeline:
@list |> map net_aimd_step |> p
```

net_alohanet_throughput

`net_alohanet_throughput` — batch 49 builtin.

```
my $result = net_alohanet_throughput $x
# or in a pipeline:
@list |> map net_alohanet_throughput |> p
```

net_alpha_fair_step

`net_alpha_fair_step` — batch 49 builtin.

```
my $result = net_alpha_fair_step $x
# or in a pipeline:
@list |> map net_alpha_fair_step |> p
```

net_amc_threshold_index

net_amc_threshold_index — batch 49 builtin.

```
my $result = net_amc_threshold_index $x
# or in a pipeline:
@list |> map net_amc_threshold_index |> p
```

net_aqm_blue_step

net_aqm_blue_step — batch 49 builtin.

```
my $result = net_aqm_blue_step $x
# or in a pipeline:
@list |> map net_aqm_blue_step |> p
```

net_aqm_choke_step

net_aqm_choke_step — batch 49 builtin.

```
my $result = net_aqm_choke_step $x
# or in a pipeline:
@list |> map net_aqm_choke_step |> p
```

net_aqm_codel_target

net_aqm_codel_target — batch 49 builtin.

```
my $result = net_aqm_codel_target $x
# or in a pipeline:
@list |> map net_aqm_codel_target |> p
```

net_aqm_drr_step

net_aqm_drr_step — batch 49 builtin.

```
my $result = net_aqm_drr_step $x
# or in a pipeline:
@list |> map net_aqm_drr_step |> p
```

net_aqm_fq_codel_step

net_aqm_fq_codel_step — batch 49 builtin.

```
my $result = net_aqm_fq_codel_step $x
# or in a pipeline:
@list |> map net_aqm_fq_codel_step |> p
```

net_aqm_pie_drop_rate

net_aqm_pie_drop_rate — batch 49 builtin.

```
my $result = net_aqm_pie_drop_rate $x
# or in a pipeline:
@list |> map net_aqm_pie_drop_rate |> p
```

net_aqm_red_drop_prob

net_aqm_red_drop_prob — batch 49 builtin.

```
my $result = net_aqm_red_drop_prob $x
# or in a pipeline:
@list |> map net_aqm_red_drop_prob |> p
```

net_aqm_sfq_step

`net_aqm_sfq_step` — batch 49 builtin.

```
my $result = net_aqm_sfq_step $x
# or in a pipeline:
@list |> map net_aqm_sfq_step |> p
```

net_aqm_wrr_step

`net_aqm_wrr_step` — batch 49 builtin.

```
my $result = net_aqm_wrr_step $x
# or in a pipeline:
@list |> map net_aqm_wrr_step |> p
```

net_array_gain

`net_array_gain` — batch 49 builtin.

```
my $result = net_array_gain $x
# or in a pipeline:
@list |> map net_array_gain |> p
```

net_backpressure_step

`net_backpressure_step` — batch 49 builtin.

```
my $result = net_backpressure_step $x
# or in a pipeline:
@list |> map net_backpressure_step |> p
```

net_bandwidth_delay_product

`net_bandwidth_delay_product` — batch 49 builtin.

```
my $result = net_bandwidth_delay_product $x
# or in a pipeline:
@list |> map net_bandwidth_delay_product |> p
```

net_bcjr_step

`net_bcjr_step` — batch 49 builtin.

```
my $result = net_bcjr_step $x
# or in a pipeline:
@list |> map net_bcjr_step |> p
```

net_blocking_probability

`net_blocking_probability` — batch 49 builtin.

```
my $result = net_blocking_probability $x
# or in a pipeline:
@list |> map net_blocking_probability |> p
```

net_burst_size_compute

`net_burst_size_compute` — batch 49 builtin.

```
my $result = net_burst_size_compute $x
# or in a pipeline:
@list |> map net_burst_size_compute |> p
```

net_call_admission_check

net_call_admission_check — batch 49 builtin.

```
my $result = net_call_admission_check $x
# or in a pipeline:
@list |> map net_call_admission_check |> p
```

net_capacity_shannon

net_capacity_shannon — batch 49 builtin.

```
my $result = net_capacity_shannon $x
# or in a pipeline:
@list |> map net_capacity_shannon |> p
```

net_coding_gain

net_coding_gain — batch 49 builtin.

```
my $result = net_coding_gain $x
# or in a pipeline:
@list |> map net_coding_gain |> p
```

net_csma_back_off

net_csma_back_off — batch 49 builtin.

```
my $result = net_csma_back_off $x
# or in a pipeline:
@list |> map net_csma_back_off |> p
```

net_csma_efficiency

net_csma_efficiency — batch 49 builtin.

```
my $result = net_csma_efficiency $x
# or in a pipeline:
@list |> map net_csma_efficiency |> p
```

net_diversity_gain

net_diversity_gain — batch 49 builtin.

```
my $result = net_diversity_gain $x
# or in a pipeline:
@list |> map net_diversity_gain |> p
```

net_doppler_shift

net_doppler_shift — batch 49 builtin.

```
my $result = net_doppler_shift $x
# or in a pipeline:
@list |> map net_doppler_shift |> p
```

net_drop_tail_check

net_drop_tail_check — batch 49 builtin.

```
my $result = net_drop_tail_check $x
# or in a pipeline:
@list |> map net_drop_tail_check |> p
```

net_engset_formula

net_engset_formula — batch 49 builtin.

```
my $result = net_engset_formula $x
# or in a pipeline:
@list |> map net_engset_formula |> p
```

net_erlang_b_formula

net_erlang_b_formula — batch 49 builtin.

```
my $result = net_erlang_b_formula $x
# or in a pipeline:
@list |> map net_erlang_b_formula |> p
```

net_erlang_c_formula

net_erlang_c_formula — batch 49 builtin.

```
my $result = net_erlang_c_formula $x
# or in a pipeline:
@list |> map net_erlang_c_formula |> p
```

net_forced_flow_law

net_forced_flow_law — batch 49 builtin.

```
my $result = net_forced_flow_law $x
# or in a pipeline:
@list |> map net_forced_flow_law |> p
```

net_friis_received_power

net_friis_received_power — batch 49 builtin.

```
my $result = net_friis_received_power $x
# or in a pipeline:
@list |> map net_friis_received_power |> p
```

net_handoff_threshold

net_handoff_threshold — batch 49 builtin.

```
my $result = net_handoff_threshold $x
# or in a pipeline:
@list |> map net_handoff_threshold |> p
```

net_harq_combining_gain

net_harq_combining_gain — batch 49 builtin.

```
my $result = net_harq_combining_gain $x
# or in a pipeline:
@list |> map net_harq_combining_gain |> p
```

net_jitter_estimate

net_jitter_estimate — batch 49 builtin.

```
my $result = net_jitter_estimate $x
# or in a pipeline:
@list |> map net_jitter_estimate |> p
```

net_kelly_pricing_step

net_kelly_pricing_step — batch 49 builtin.

```
my $result = net_kelly_pricing_step $x
# or in a pipeline:
@list |> map net_kelly_pricing_step |> p
```

net_latency_avg

net_latency_avg — batch 49 builtin.

```
my $result = net_latency_avg $x
# or in a pipeline:
@list |> map net_latency_avg |> p
```

net_ldpc_iteration_step

net_ldpc_iteration_step — batch 49 builtin.

```
my $result = net_ldpc_iteration_step $x
# or in a pipeline:
@list |> map net_ldpc_iteration_step |> p
```

net_link_capacity_share

net_link_capacity_share — batch 49 builtin.

```
my $result = net_link_capacity_share $x
# or in a pipeline:
@list |> map net_link_capacity_share |> p
```

net_little_law_l

net_little_law_l — batch 49 builtin.

```
my $result = net_little_law_l $x
# or in a pipeline:
@list |> map net_little_law_l |> p
```

net_log_distance_path

net_log_distance_path — batch 49 builtin.

```
my $result = net_log_distance_path $x
# or in a pipeline:
@list |> map net_log_distance_path |> p
```

net_loss_rate_to_throughput

net_loss_rate_to_throughput — batch 49 builtin.

```
my $result = net_loss_rate_to_throughput $input
```

net_lyapunov_drift_plus_penalty

net_lyapunov_drift_plus_penalty — batch 49 builtin.

```
my $result = net_lyapunov_drift_plus_penalty $x
# or in a pipeline:
@list |> map net_lyapunov_drift_plus_penalty |> p
```

net_macro_diversity_step

net_macro_diversity_step — batch 49 builtin.

```
my $result = net_macro_diversity_step $x
# or in a pipeline:
@list |> map net_macro_diversity_step |> p
```

net_max_min_fair_step

net_max_min_fair_step — batch 49 builtin.

```
my $result = net_max_min_fair_step $x
# or in a pipeline:
@list |> map net_max_min_fair_step |> p
```

net_max_weight_match

net_max_weight_match — batch 49 builtin.

```
my $result = net_max_weight_match $x
# or in a pipeline:
@list |> map net_max_weight_match |> p
```

net_miad_step

net_miad_step — batch 49 builtin.

```
my $result = net_miad_step $x
# or in a pipeline:
@list |> map net_miad_step |> p
```

net_micro_diversity_step

net_micro_diversity_step — batch 49 builtin.

```
my $result = net_micro_diversity_step $x
# or in a pipeline:
@list |> map net_micro_diversity_step |> p
```

net_mimd_step

net_mimd_step — batch 49 builtin.

```
my $result = net_mimd_step $x
# or in a pipeline:
@list |> map net_mimd_step |> p
```

net_mimo_capacity_step

net_mimo_capacity_step — batch 49 builtin.

```
my $result = net_mimo_capacity_step $x
# or in a pipeline:
@list |> map net_mimo_capacity_step |> p
```

net_mmse_beam_step

net_mmse_beam_step — batch 49 builtin.

```
my $result = net_mmse_beam_step $x
# or in a pipeline:
@list |> map net_mmse_beam_step |> p
```

net_multiplexing_gain

net_multiplexing_gain — batch 49 builtin.

```
my $result = net_multiplexing_gain $x
# or in a pipeline:
@list |> map net_multiplexing_gain |> p
```

net_network_utility_max

net_network_utility_max — batch 49 builtin.

```
my $result = net_network_utility_max $x
# or in a pipeline:
@list |> map net_network_utility_max |> p
```

net_okumura_hata_loss

net_okumura_hata_loss — batch 49 builtin.

```
my $result = net_okumura_hata_loss $x
# or in a pipeline:
@list |> map net_okumura_hata_loss |> p
```

net_outage_probability

net_outage_probability — batch 49 builtin.

```
my $result = net_outage_probability $x
# or in a pipeline:
@list |> map net_outage_probability |> p
```

net_packet_loss_estimate

net_packet_loss_estimate — batch 49 builtin.

```
my $result = net_packet_loss_estimate $x
# or in a pipeline:
@list |> map net_packet_loss_estimate |> p
```

net_packet_pacing_step

net_packet_pacing_step — batch 49 builtin.

```
my $result = net_packet_pacing_step $x
# or in a pipeline:
@list |> map net_packet_pacing_step |> p
```

net_path_capacity_kleinrock

net_path_capacity_kleinrock — batch 49 builtin.

```
my $result = net_path_capacity_kleinrock $x
# or in a pipeline:
@list |> map net_path_capacity_kleinrock |> p
```

net_polar_decode_step

net_polar_decode_step — batch 49 builtin.

```
my $result = net_polar_decode_step $x
# or in a pipeline:
@list |> map net_polar_decode_step |> p
```

net_polling_efficiency

net_polling_efficiency — batch 49 builtin.

```
my $result = net_polling_efficiency $x
# or in a pipeline:
@list |> map net_polling_efficiency |> p
```

net_priority_queue_index

net_priority_queue_index — batch 49 builtin.

```
my $result = net_priority_queue_index $x
# or in a pipeline:
@list |> map net_priority_queue_index |> p
```

net_proportional_fair_share

net_proportional_fair_share — batch 49 builtin.

```
my $result = net_proportional_fair_share $x
# or in a pipeline:
@list |> map net_proportional_fair_share |> p
```

net_pruning_gain

net_pruning_gain — batch 49 builtin.

```
my $result = net_pruning_gain $x
# or in a pipeline:
@list |> map net_pruning_gain |> p
```

net_qcsma_propose

net_qcsma_propose — batch 49 builtin.

```
my $result = net_qcsma_propose $x
# or in a pipeline:
@list |> map net_qcsma_propose |> p
```

net_radio_path_loss

net_radio_path_loss — batch 49 builtin.

```
my $result = net_radio_path_loss $x
# or in a pipeline:
@list |> map net_radio_path_loss |> p
```

net_rayleigh_envelope

net_rayleigh_envelope — batch 49 builtin.

```
my $result = net_rayleigh_envelope $x
# or in a pipeline:
@list |> map net_rayleigh_envelope |> p
```

net_reproductive_rate

net_reproductive_rate — batch 22 builtin.

```
my $result = net_reproductive_rate $x
# or in a pipeline:
@list |> map net_reproductive_rate |> p
```

net_response_time_law

`net_response_time_law` — batch 49 builtin.

```
my $result = net_response_time_law $x
# or in a pipeline:
@list |> map net_response_time_law |> p
```

net_rician_k_factor

`net_rician_k_factor` — batch 49 builtin.

```
my $result = net_rician_k_factor $x
# or in a pipeline:
@list |> map net_rician_k_factor |> p
```

net_router_buffer_size

`net_router_buffer_size` — batch 49 builtin.

```
my $result = net_router_buffer_size $x
# or in a pipeline:
@list |> map net_router_buffer_size |> p
```

net_rto_compute

`net_rto_compute` — batch 49 builtin.

```
my $result = net_rto_compute $input
```

net_rtt_smoothed

`net_rtt_smoothed` — batch 49 builtin.

```
my $result = net_rtt_smoothed $x
# or in a pipeline:
@list |> map net_rtt_smoothed |> p
```

net_rtt_variation

`net_rtt_variation` — batch 49 builtin.

```
my $result = net_rtt_variation $x
# or in a pipeline:
@list |> map net_rtt_variation |> p
```

net_shadowing_normal

`net_shadowing_normal` — batch 49 builtin.

```
my $result = net_shadowing_normal $x
# or in a pipeline:
@list |> map net_shadowing_normal |> p
```

net_slotted_aloha_throughput

`net_slotted_aloha_throughput` — batch 49 builtin.

```
my $result = net_slotted_aloha_throughput $x
# or in a pipeline:
@list |> map net_slotted_aloha_throughput |> p
```

net_tcp_bbr_step

net_tcp_bbr_step — batch 49 builtin.

```
my $result = net_tcp_bbr_step $x
# or in a pipeline:
@list |> map net_tcp_bbr_step |> p
```

net_tcp_compound_step

net_tcp_compound_step — batch 49 builtin.

```
my $result = net_tcp_compound_step $x
# or in a pipeline:
@list |> map net_tcp_compound_step |> p
```

net_tcp_cubic_step

net_tcp_cubic_step — batch 49 builtin.

```
my $result = net_tcp_cubic_step $x
# or in a pipeline:
@list |> map net_tcp_cubic_step |> p
```

net_tcp_cwnd_step

net_tcp_cwnd_step — batch 49 builtin.

```
my $result = net_tcp_cwnd_step $x
# or in a pipeline:
@list |> map net_tcp_cwnd_step |> p
```

net_tcp_dctcp_step

net_tcp_dctcp_step — batch 49 builtin.

```
my $result = net_tcp_dctcp_step $x
# or in a pipeline:
@list |> map net_tcp_dctcp_step |> p
```

net_tcp_htcp_step

net_tcp_htcp_step — batch 49 builtin.

```
my $result = net_tcp_htcp_step $x
# or in a pipeline:
@list |> map net_tcp_htcp_step |> p
```

net_tcp_hybla_step

net_tcp_hybla_step — batch 49 builtin.

```
my $result = net_tcp_hybla_step $x
# or in a pipeline:
@list |> map net_tcp_hybla_step |> p
```

net_tcp_illinois_step

net_tcp_illinois_step — batch 49 builtin.

```
my $result = net_tcp_illinois_step $x
# or in a pipeline:
@list |> map net_tcp_illinois_step |> p
```

net_tcp_lp_step

net_tcp_lp_step — batch 49 builtin.

```
my $result = net_tcp_lp_step $x
# or in a pipeline:
@list |> map net_tcp_lp_step |> p
```

net_tcp_reno_step

net_tcp_reno_step — batch 49 builtin.

```
my $result = net_tcp_reno_step $x
# or in a pipeline:
@list |> map net_tcp_reno_step |> p
```

net_tcp_scalable_step

net_tcp_scalable_step — batch 49 builtin.

```
my $result = net_tcp_scalable_step $x
# or in a pipeline:
@list |> map net_tcp_scalable_step |> p
```

net_tcp_ssthresh_update

net_tcp_ssthresh_update — batch 49 builtin.

```
my $result = net_tcp_ssthresh_update $x
# or in a pipeline:
@list |> map net_tcp_ssthresh_update |> p
```

net_tcp_vegas_step

net_tcp_vegas_step — batch 49 builtin.

```
my $result = net_tcp_vegas_step $x
# or in a pipeline:
@list |> map net_tcp_vegas_step |> p
```

net_tcp_veno_step

net_tcp_veno_step — batch 49 builtin.

```
my $result = net_tcp_veno_step $x
# or in a pipeline:
@list |> map net_tcp_veno_step |> p
```

net_tcp_westwood_step

net_tcp_westwood_step — batch 49 builtin.

```
my $result = net_tcp_westwood_step $x
# or in a pipeline:
@list |> map net_tcp_westwood_step |> p
```

net_tcp_yeah_step

net_tcp_yeah_step — batch 49 builtin.

```
my $result = net_tcp_yeah_step $x
# or in a pipeline:
@list |> map net_tcp_yeah_step |> p
```

net_throughput_law

`net_throughput_law` — batch 49 builtin.

```
my $result = net_throughput_law $x
# or in a pipeline:
@list |> map net_throughput_law |> p
```

net_throughput_mathis

`net_throughput_mathis` — batch 49 builtin.

```
my $result = net_throughput_mathis $x
# or in a pipeline:
@list |> map net_throughput_mathis |> p
```

net_throughput_padhye

`net_throughput_padhye` — batch 49 builtin.

```
my $result = net_throughput_padhye $x
# or in a pipeline:
@list |> map net_throughput_padhye |> p
```

net_throughput_response

`net_throughput_response` — batch 49 builtin.

```
my $result = net_throughput_response $x
# or in a pipeline:
@list |> map net_throughput_response |> p
```

net_token_rate_limit

`net_token_rate_limit` — batch 49 builtin.

```
my $result = net_token_rate_limit $x
# or in a pipeline:
@list |> map net_token_rate_limit |> p
```

net_token_ring_efficiency

`net_token_ring_efficiency` — batch 49 builtin.

```
my $result = net_token_ring_efficiency $x
# or in a pipeline:
@list |> map net_token_ring_efficiency |> p
```

net_traffic_shaper_step

`net_traffic_shaper_step` — batch 49 builtin.

```
my $result = net_traffic_shaper_step $x
# or in a pipeline:
@list |> map net_traffic_shaper_step |> p
```

net_turbo_decode_iter

`net_turbo_decode_iter` — batch 49 builtin.

```
my $result = net_turbo_decode_iter $x
# or in a pipeline:
@list |> map net_turbo_decode_iter |> p
```

net_two_ray_ground_loss

`net_two_ray_ground_loss` — batch 49 builtin.

```
my $result = net_two_ray_ground_loss $x
# or in a pipeline:
@list |> map net_two_ray_ground_loss |> p
```

net_utilization_law

`net_utilization_law` — batch 49 builtin.

```
my $result = net_utilization_law $x
# or in a pipeline:
@list |> map net_utilization_law |> p
```

net_viterbi_step

`net_viterbi_step` — batch 49 builtin.

```
my $result = net_viterbi_step $x
# or in a pipeline:
@list |> map net_viterbi_step |> p
```

net_water_filling_power

`net_water_filling_power` — batch 49 builtin.

```
my $result = net_water_filling_power $x
# or in a pipeline:
@list |> map net_water_filling_power |> p
```

net_zero_forcing_beam

`net_zero_forcing_beam` — batch 49 builtin.

```
my $result = net_zero_forcing_beam $x
# or in a pipeline:
@list |> map net_zero_forcing_beam |> p
```

network_formation_step

`network_formation_step` — batch 43 builtin.

```
my $result = network_formation_step $x
# or in a pipeline:
@list |> map network_formation_step |> p
```

neutron_mass

`neutron_mass` — physics constants builtin.

```
my $result = neutron_mass $x
# or in a pipeline:
@list |> map neutron_mass |> p
```

newey_west_se

`newey_west_se` — batch 36 builtin.

```
my $result = newey_west_se $x
# or in a pipeline:
@list |> map newey_west_se |> p
```

newmark_beta_step

`newmark_beta_step` — batch 34 builtin.

```
my $result = newmark_beta_step $x
# or in a pipeline:
@list |> map newmark_beta_step |> p
```

newton

`newton_method` (aliases `newton`, `newton_raphson`) finds a root via Newton-Raphson. Takes f, f', x0, and optional tolerance.

```
my $root = newton(fn { $_[0]**2 - 2 }, fn { 2*$_[0] }, 1.5)  # √2
```

newtons_to_lbf

`newtons_to_lbf` — file stat / path builtin.

```
my $result = newtons_to_lbf $input
```

next_permutation

`next_permutation` — matrix operations (uncategorized batch) builtin.

```
my $result = next_permutation $x
# or in a pipeline:
@list |> map next_permutation |> p
```

next_prime

`next_prime` — number theory / primes builtin.

```
my $result = next_prime $x
# or in a pipeline:
@list |> map next_prime |> p
```

nfa_to_dfa

`nfa_to_dfa` — batch 68 builtin.

```
my $result = nfa_to_dfa $input
```

nfirst

`nfirst` — algebraic match builtin.

```
my $result = nfirst $x
# or in a pipeline:
@list |> map nfirst |> p
```

nfs

`nfs` — algebraic match builtin. Alias for `nfirst`.

```
my $result = nfs $x
# or in a pipeline:
@list |> map nfs |> p
```

ngram

`ngram` — string processing (uncategorized batch) builtin. Alias for `ngrams`.

```
my $result = ngram $x
# or in a pipeline:
@list |> map ngram |> p
```

ngrams

`ngrams` — string processing (uncategorized batch) builtin.

```
my $result = ngrams $x
# or in a pipeline:
@list |> map ngrams |> p
```

nichols_chart_step

`nichols_chart_step` — batch 59 builtin.

```
my $result = nichols_chart_step $x
# or in a pipeline:
@list |> map nichols_chart_step |> p
```

nilpotency_class_lower

`nilpotency_class_lower` — batch 37 builtin.

```
my $result = nilpotency_class_lower $x
# or in a pipeline:
@list |> map nilpotency_class_lower |> p
```

nlp_alibi_position_bias

`nlp_alibi_position_bias` — batch 46 builtin.

```
my $result = nlp_alibi_position_bias $x
# or in a pipeline:
@list |> map nlp_alibi_position_bias |> p
```

nlp_attention_pool_step

`nlp_attention_pool_step` — batch 46 builtin.

```
my $result = nlp_attention_pool_step $x
# or in a pipeline:
@list |> map nlp_attention_pool_step |> p
```

nlp_avg_pool_step

`nlp_avg_pool_step` — batch 46 builtin.

```
my $result = nlp_avg_pool_step $x
# or in a pipeline:
@list |> map nlp_avg_pool_step |> p
```

nlp_bigbird_step

`nlp_bigbird_step` — batch 46 builtin.

```
my $result = nlp_bigbird_step $x
# or in a pipeline:
@list |> map nlp_bigbird_step |> p
```

nlp_bits_per_character

`nlp_bits_per_character` — batch 46 builtin.

```
my $result = nlp_bits_per_character $x
# or in a pipeline:
@list |> map nlp_bits_per_character |> p
```

nlp_bleu_score_n

`nlp_bleu_score_n` — batch 46 builtin.

```
my $result = nlp_bleu_score_n $x
# or in a pipeline:
@list |> map nlp_bleu_score_n |> p
```

nlp_block_attn_step

`nlp_block_attn_step` — batch 46 builtin.

```
my $result = nlp_block_attn_step $x
# or in a pipeline:
@list |> map nlp_block_attn_step |> p
```

nlp_bm25_score

`nlp_bm25_score` — batch 46 builtin.

```
my $result = nlp_bm25_score $x
# or in a pipeline:
@list |> map nlp_bm25_score |> p
```

nlp_byte_frequency

`nlp_byte_frequency` — batch 46 builtin.

```
my $result = nlp_byte_frequency $x
# or in a pipeline:
@list |> map nlp_byte_frequency |> p
```

nlp_byte_level_bpe_step

`nlp_byte_level_bpe_step` — batch 46 builtin.

```
my $result = nlp_byte_level_bpe_step $x
# or in a pipeline:
@list |> map nlp_byte_level_bpe_step |> p
```

nlp_byte_pair_merge_step

`nlp_byte_pair_merge_step` — batch 46 builtin.

```
my $result = nlp_byte_pair_merge_step $x
# or in a pipeline:
@list |> map nlp_byte_pair_merge_step |> p
```

nlp_cer_score

`nlp_cer_score` — batch 46 builtin.

```
my $result = nlp_cer_score $x
# or in a pipeline:
@list |> map nlp_cer_score |> p
```

nlp_char_frequency

`nlp_char_frequency` — batch 46 builtin.

```
my $result = nlp_char_frequency $x
# or in a pipeline:
@list |> map nlp_char_frequency |> p
```

nlp_char_ngram_count

`nlp_char_ngram_count` — batch 46 builtin.

```
my $result = nlp_char_ngram_count $x
# or in a pipeline:
@list |> map nlp_char_ngram_count |> p
```

nlp_chi2_collocation

`nlp_chi2_collocation` — batch 46 builtin.

```
my $result = nlp_chi2_collocation $x
# or in a pipeline:
@list |> map nlp_chi2_collocation |> p
```

nlp_chrf_score

`nlp_chrf_score` — batch 46 builtin.

```
my $result = nlp_chrf_score $x
# or in a pipeline:
@list |> map nlp_chrf_score |> p
```

nlp_cosine_similarity_two

`nlp_cosine_similarity_two` — batch 46 builtin.

```
my $result = nlp_cosine_similarity_two $x
# or in a pipeline:
@list |> map nlp_cosine_similarity_two |> p
```

nlp_cross_attn_compute_step

`nlp_cross_attn_compute_step` — batch 46 builtin.

```
my $result = nlp_cross_attn_compute_step $x
# or in a pipeline:
@list |> map nlp_cross_attn_compute_step |> p
```

nlp_damerau_levenshtein

`nlp_damerau_levenshtein` — batch 46 builtin.

```
my $result = nlp_damerau_levenshtein $x
# or in a pipeline:
@list |> map nlp_damerau_levenshtein |> p
```

nlp_dice_coefficient_two

nlp_dice_coefficient_two — batch 46 builtin.

```
my $result = nlp_dice_coefficient_two $x
# or in a pipeline:
@list |> map nlp_dice_coefficient_two |> p
```

nlp_digit_ratio

nlp_digit_ratio — batch 46 builtin.

```
my $result = nlp_digit_ratio $x
# or in a pipeline:
@list |> map nlp_digit_ratio |> p
```

nlp_dilated_attn_step

nlp_dilated_attn_step — batch 46 builtin.

```
my $result = nlp_dilated_attn_step $x
# or in a pipeline:
@list |> map nlp_dilated_attn_step |> p
```

nlp_dirichlet_smoothing

nlp_dirichlet_smoothing — batch 46 builtin.

```
my $result = nlp_dirichlet_smoothing $x
# or in a pipeline:
@list |> map nlp_dirichlet_smoothing |> p
```

nlp_doc_embedding_avg

nlp_doc_embedding_avg — batch 46 builtin.

```
my $result = nlp_doc_embedding_avg $x
# or in a pipeline:
@list |> map nlp_doc_embedding_avg |> p
```

nlp_doc_freq_step

nlp_doc_freq_step — batch 46 builtin.

```
my $result = nlp_doc_freq_step $x
# or in a pipeline:
@list |> map nlp_doc_freq_step |> p
```

nlp_dunning_log_likelihood

nlp_dunning_log_likelihood — batch 46 builtin.

```
my $result = nlp_dunning_log_likelihood $x
# or in a pipeline:
@list |> map nlp_dunning_log_likelihood |> p
```

nlp_email_count

nlp_email_count — batch 46 builtin.

```
my $result = nlp_email_count $x
# or in a pipeline:
@list |> map nlp_email_count |> p
```

nlp_emoji_ratio

nlp_emoji_ratio — batch 46 builtin.

```
my $result = nlp_emoji_ratio $x
# or in a pipeline:
@list |> map nlp_emoji_ratio |> p
```

nlp_fasttext_subword_count

nlp_fasttext_subword_count — batch 46 builtin.

```
my $result = nlp_fasttext_subword_count $x
# or in a pipeline:
@list |> map nlp_fasttext_subword_count |> p
```

nlp_global_attn_step

nlp_global_attn_step — batch 46 builtin.

```
my $result = nlp_global_attn_step $x
# or in a pipeline:
@list |> map nlp_global_attn_step |> p
```

nlp_glove_loss_step

nlp_glove_loss_step — batch 46 builtin.

```
my $result = nlp_glove_loss_step $x
# or in a pipeline:
@list |> map nlp_glove_loss_step |> p
```

nlp_good_turing_count

nlp_good_turing_count — batch 46 builtin.

```
my $result = nlp_good_turing_count $x
# or in a pipeline:
@list |> map nlp_good_turing_count |> p
```

nlp_hamming_distance

nlp_hamming_distance — batch 46 builtin.

```
my $result = nlp_hamming_distance $x
# or in a pipeline:
@list |> map nlp_hamming_distance |> p
```

nlp_hashtag_count

nlp_hashtag_count — batch 46 builtin.

```
my $result = nlp_hashtag_count $x
# or in a pipeline:
@list |> map nlp_hashtag_count |> p
```

nlp_inverse_doc_freq

nlp_inverse_doc_freq — batch 46 builtin.

```
my $result = nlp_inverse_doc_freq $x
# or in a pipeline:
@list |> map nlp_inverse_doc_freq |> p
```

nlp_jaccard_similarity_two

nlp_jaccard_similarity_two — batch 46 builtin.

```
my $result = nlp_jaccard_similarity_two $x
# or in a pipeline:
@list |> map nlp_jaccard_similarity_two |> p
```

nlp_jaro_distance

nlp_jaro_distance — batch 46 builtin.

```
my $result = nlp_jaro_distance $x
# or in a pipeline:
@list |> map nlp_jaro_distance |> p
```

nlp_jaro_winkler

nlp_jaro_winkler — batch 46 builtin.

```
my $result = nlp_jaro_winkler $x
# or in a pipeline:
@list |> map nlp_jaro_winkler |> p
```

nlp_jelinek_mercer

nlp_jelinek_mercer — batch 46 builtin.

```
my $result = nlp_jelinek_mercer $x
# or in a pipeline:
@list |> map nlp_jelinek_mercer |> p
```

nlp_kl_lm_div

nlp_kl_lm_div — batch 46 builtin.

```
my $result = nlp_kl_lm_div $x
# or in a pipeline:
@list |> map nlp_kl_lm_div |> p
```

nlp_kneser_ney_step

nlp_kneser_ney_step — batch 46 builtin.

```
my $result = nlp_kneser_ney_step $x
# or in a pipeline:
@list |> map nlp_kneser_ney_step |> p
```

nlp_laplace_smoothing

nlp_laplace_smoothing — batch 46 builtin.

```
my $result = nlp_laplace_smoothing $x
# or in a pipeline:
@list |> map nlp_laplace_smoothing |> p
```

nlp_lcs_length

nlp_lcs_length — batch 46 builtin.

```
my $result = nlp_lcs_length $x
# or in a pipeline:
@list |> map nlp_lcs_length |> p
```

nlp_lcs_ratio

`nlp_lcs_ratio` — batch 46 builtin.

```
my $result = nlp_lcs_ratio $x
# or in a pipeline:
@list |> map nlp_lcs_ratio |> p
```

nlp_lda_alpha_step

`nlp_lda_alpha_step` — batch 46 builtin.

```
my $result = nlp_lda_alpha_step $x
# or in a pipeline:
@list |> map nlp_lda_alpha_step |> p
```

nlp_lda_beta_step

`nlp_lda_beta_step` — batch 46 builtin.

```
my $result = nlp_lda_beta_step $x
# or in a pipeline:
@list |> map nlp_lda_beta_step |> p
```

nlp_lda_topic_dist

`nlp_lda_topic_dist` — batch 46 builtin.

```
my $result = nlp_lda_topic_dist $x
# or in a pipeline:
@list |> map nlp_lda_topic_dist |> p
```

nlp_levenshtein_dist

`nlp_levenshtein_dist` — batch 46 builtin.

```
my $result = nlp_levenshtein_dist $x
# or in a pipeline:
@list |> map nlp_levenshtein_dist |> p
```

nlp_lidstone_smoothing

`nlp_lidstone_smoothing` — batch 46 builtin.

```
my $result = nlp_lidstone_smoothing $x
# or in a pipeline:
@list |> map nlp_lidstone_smoothing |> p
```

nlp_linformer_step

`nlp_linformer_step` — batch 46 builtin.

```
my $result = nlp_linformer_step $x
# or in a pipeline:
@list |> map nlp_linformer_step |> p
```

nlp_local_attn_step

`nlp_local_attn_step` — batch 46 builtin.

```
my $result = nlp_local_attn_step $x
# or in a pipeline:
@list |> map nlp_local_attn_step |> p
```

nlp_loglikelihood_collocation

nlp_loglikelihood_collocation — batch 46 builtin.

```
my $result = nlp_loglikelihood_collocation $x
# or in a pipeline:
@list |> map nlp_loglikelihood_collocation |> p
```

nlp_longformer_step

nlp_longformer_step — batch 46 builtin.

```
my $result = nlp_longformer_step $x
# or in a pipeline:
@list |> map nlp_longformer_step |> p
```

nlp_max_pool_step

nlp_max_pool_step — batch 46 builtin.

```
my $result = nlp_max_pool_step $x
# or in a pipeline:
@list |> map nlp_max_pool_step |> p
```

nlp_max_seq_len_check

nlp_max_seq_len_check — batch 46 builtin.

```
my $result = nlp_max_seq_len_check $x
# or in a pipeline:
@list |> map nlp_max_seq_len_check |> p
```

nlp_mention_count

nlp_mention_count — batch 46 builtin.

```
my $result = nlp_mention_count $x
# or in a pipeline:
@list |> map nlp_mention_count |> p
```

nlp_meteor_score

nlp_meteor_score — batch 46 builtin.

```
my $result = nlp_meteor_score $x
# or in a pipeline:
@list |> map nlp_meteor_score |> p
```

nlp_npmi_score

nlp_npmi_score — batch 46 builtin.

```
my $result = nlp_npmi_score $x
# or in a pipeline:
@list |> map nlp_npmi_score |> p
```

nlp_okapi_score

nlp_okapi_score — batch 46 builtin.

```
my $result = nlp_okapi_score $x
# or in a pipeline:
@list |> map nlp_okapi_score |> p
```

nlp_overlap_coefficient

`nlp_overlap_coefficient` — batch 46 builtin.

```
my $result = nlp_overlap_coefficient $x
# or in a pipeline:
@list |> map nlp_overlap_coefficient |> p
```

nlp_pe_freq_band

`nlp_pe_freq_band` — batch 46 builtin.

```
my $result = nlp_pe_freq_band $x
# or in a pipeline:
@list |> map nlp_pe_freq_band |> p
```

nlp_performer_step

`nlp_performer_step` — batch 46 builtin.

```
my $result = nlp_performer_step $x
# or in a pipeline:
@list |> map nlp_performer_step |> p
```

nlp_perplexity_value

`nlp_perplexity_value` — batch 46 builtin.

```
my $result = nlp_perplexity_value $x
# or in a pipeline:
@list |> map nlp_perplexity_value |> p
```

nlp_phone_count

`nlp_phone_count` — batch 46 builtin.

```
my $result = nlp_phone_count $x
# or in a pipeline:
@list |> map nlp_phone_count |> p
```

nlp_plsa_step

`nlp_plsa_step` — batch 46 builtin.

```
my $result = nlp_plsa_step $x
# or in a pipeline:
@list |> map nlp_plsa_step |> p
```

nlp_pmi_score

`nlp_pmi_score` — batch 46 builtin.

```
my $result = nlp_pmi_score $x
# or in a pipeline:
@list |> map nlp_pmi_score |> p
```

nlp_pointwise_attn_score

`nlp_pointwise_attn_score` — batch 46 builtin.

```
my $result = nlp_pointwise_attn_score $x
# or in a pipeline:
@list |> map nlp_pointwise_attn_score |> p
```

nlp_position_encoding_cos

nlp_position_encoding_cos — batch 46 builtin.

```
my $result = nlp_position_encoding_cos $x
# or in a pipeline:
@list |> map nlp_position_encoding_cos |> p
```

nlp_position_encoding_sin

nlp_position_encoding_sin — batch 46 builtin.

```
my $result = nlp_position_encoding_sin $x
# or in a pipeline:
@list |> map nlp_position_encoding_sin |> p
```

nlp_punct_ratio

nlp_punct_ratio — batch 46 builtin.

```
my $result = nlp_punct_ratio $x
# or in a pipeline:
@list |> map nlp_punct_ratio |> p
```

nlp_query_likelihood_step

nlp_query_likelihood_step — batch 46 builtin.

```
my $result = nlp_query_likelihood_step $x
# or in a pipeline:
@list |> map nlp_query_likelihood_step |> p
```

nlp_reformer_step

nlp_reformer_step — batch 46 builtin.

```
my $result = nlp_reformer_step $x
# or in a pipeline:
@list |> map nlp_reformer_step |> p
```

nlp_relative_position_bias

nlp_relative_position_bias — batch 46 builtin.

```
my $result = nlp_relative_position_bias $x
# or in a pipeline:
@list |> map nlp_relative_position_bias |> p
```

nlp_rope_apply_step

nlp_rope_apply_step — batch 46 builtin.

```
my $result = nlp_rope_apply_step $x
# or in a pipeline:
@list |> map nlp_rope_apply_step |> p
```

nlp_rope_rotary_angle

nlp_rope_rotary_angle — batch 46 builtin.

```
my $result = nlp_rope_rotary_angle $x
# or in a pipeline:
@list |> map nlp_rope_rotary_angle |> p
```

nlp_rouge_score_n

nlp_rouge_score_n — batch 46 builtin.

```
my $result = nlp_rouge_score_n $x
# or in a pipeline:
@list |> map nlp_rouge_score_n |> p
```

nlp_routing_attn_step

nlp_routing_attn_step — batch 46 builtin.

```
my $result = nlp_routing_attn_step $x
# or in a pipeline:
@list |> map nlp_routing_attn_step |> p
```

nlp_self_attn_compute_step

nlp_self_attn_compute_step — batch 46 builtin.

```
my $result = nlp_self_attn_compute_step $x
# or in a pipeline:
@list |> map nlp_self_attn_compute_step |> p
```

nlp_sentencepiece_score

nlp_sentencepiece_score — batch 46 builtin.

```
my $result = nlp_sentencepiece_score $x
# or in a pipeline:
@list |> map nlp_sentencepiece_score |> p
```

nlp_sif_weight_step

nlp_sif_weight_step — batch 46 builtin.

```
my $result = nlp_sif_weight_step $x
# or in a pipeline:
@list |> map nlp_sif_weight_step |> p
```

nlp_simpson_coefficient

nlp_simpson_coefficient — batch 46 builtin.

```
my $result = nlp_simpson_coefficient $x
# or in a pipeline:
@list |> map nlp_simpson_coefficient |> p
```

nlp_skip_gram_count

nlp_skip_gram_count — batch 46 builtin.

```
my $result = nlp_skip_gram_count $x
# or in a pipeline:
@list |> map nlp_skip_gram_count |> p
```

nlp_sliding_window_step

nlp_sliding_window_step — batch 46 builtin.

```
my $result = nlp_sliding_window_step $x
# or in a pipeline:
@list |> map nlp_sliding_window_step |> p
```

nlp_sparse_attn_score

`nlp_sparse_attn_score` — batch 46 builtin.

```
my $result = nlp_sparse_attn_score $x
# or in a pipeline:
@list |> map nlp_sparse_attn_score |> p
```

nlp_strided_attn_step

`nlp_strided_attn_step` — batch 46 builtin.

```
my $result = nlp_strided_attn_step $x
# or in a pipeline:
@list |> map nlp_strided_attn_step |> p
```

nlp_subword_regularization

`nlp_subword_regularization` — batch 46 builtin.

```
my $result = nlp_subword_regularization $x
# or in a pipeline:
@list |> map nlp_subword_regularization |> p
```

nlp_sum_pool_step

`nlp_sum_pool_step` — batch 46 builtin.

```
my $result = nlp_sum_pool_step $x
# or in a pipeline:
@list |> map nlp_sum_pool_step |> p
```

nlp_t_score_collocation

`nlp_t_score_collocation` — batch 46 builtin.

```
my $result = nlp_t_score_collocation $x
# or in a pipeline:
@list |> map nlp_t_score_collocation |> p
```

nlp_ter_score

`nlp_ter_score` — batch 46 builtin.

```
my $result = nlp_ter_score $x
# or in a pipeline:
@list |> map nlp_ter_score |> p
```

nlp_tf_idf_step

`nlp_tf_idf_step` — batch 46 builtin.

```
my $result = nlp_tf_idf_step $x
# or in a pipeline:
@list |> map nlp_tf_idf_step |> p
```

nlp_token_drop_rate

`nlp_token_drop_rate` — batch 46 builtin.

```
my $result = nlp_token_drop_rate $x
# or in a pipeline:
@list |> map nlp_token_drop_rate |> p
```

nlp_token_overlap_two

`nlp_token_overlap_two` — batch 46 builtin.

```
my $result = nlp_token_overlap_two $x
# or in a pipeline:
@list |> map nlp_token_overlap_two |> p
```

nlp_unigram_lm_score

`nlp_unigram_lm_score` — batch 46 builtin.

```
my $result = nlp_unigram_lm_score $x
# or in a pipeline:
@list |> map nlp_unigram_lm_score |> p
```

nlp_unigram_subword_loss

`nlp_unigram_subword_loss` — batch 46 builtin.

```
my $result = nlp_unigram_subword_loss $x
# or in a pipeline:
@list |> map nlp_unigram_subword_loss |> p
```

nlp_uppercase_ratio

`nlp_uppercase_ratio` — batch 46 builtin.

```
my $result = nlp_uppercase_ratio $x
# or in a pipeline:
@list |> map nlp_uppercase_ratio |> p
```

nlp_url_count

`nlp_url_count` — batch 46 builtin.

```
my $result = nlp_url_count $x
# or in a pipeline:
@list |> map nlp_url_count |> p
```

nlp_wer_score

`nlp_wer_score` — batch 46 builtin.

```
my $result = nlp_wer_score $x
# or in a pipeline:
@list |> map nlp_wer_score |> p
```

nlp_window_attn_step

`nlp_window_attn_step` — batch 46 builtin.

```
my $result = nlp_window_attn_step $x
# or in a pipeline:
@list |> map nlp_window_attn_step |> p
```

nlp_witten_bell_step

`nlp_witten_bell_step` — batch 46 builtin.

```
my $result = nlp_witten_bell_step $x
# or in a pipeline:
@list |> map nlp_witten_bell_step |> p
```

nlp_word2vec_cbow_loss

nlp_word2vec_cbow_loss — batch 46 builtin.

```
my $result = nlp_word2vec_cbow_loss $x
# or in a pipeline:
@list |> map nlp_word2vec_cbow_loss |> p
```

nlp_word2vec_skipgram_loss

nlp_word2vec_skipgram_loss — batch 46 builtin.

```
my $result = nlp_word2vec_skipgram_loss $x
# or in a pipeline:
@list |> map nlp_word2vec_skipgram_loss |> p
```

nlp_word_freq_value

nlp_word_freq_value — batch 46 builtin.

```
my $result = nlp_word_freq_value $x
# or in a pipeline:
@list |> map nlp_word_freq_value |> p
```

nlp_word_mover_dist

nlp_word_mover_dist — batch 46 builtin.

```
my $result = nlp_word_mover_dist $x
# or in a pipeline:
@list |> map nlp_word_mover_dist |> p
```

nlp_word_ngram_count

nlp_word_ngram_count — batch 46 builtin.

```
my $result = nlp_word_ngram_count $x
# or in a pipeline:
@list |> map nlp_word_ngram_count |> p
```

nlp_wordpiece_score

nlp_wordpiece_score — batch 46 builtin.

```
my $result = nlp_wordpiece_score $x
# or in a pipeline:
@list |> map nlp_wordpiece_score |> p
```

nm_from_fst

nm_from_fst — batch 33 builtin.

```
my $result = nm_from_fst $x
# or in a pipeline:
@list |> map nm_from_fst |> p
```

nmass

nmass — physics constants builtin. Alias for neutron_mass.

```
my $result = nmass $x
# or in a pipeline:
@list |> map nmass |> p
```

nne

`nne` — algebraic match builtin. Alias for `nnext`.

```
my $result = nne $x
# or in a pipeline:
@list |> map nne |> p
```

nnext

`nnext` — algebraic match builtin.

```
my $result = nnext $x
# or in a pipeline:
@list |> map nnext |> p
```

nohtml

`nohtml` — string processing (uncategorized batch) builtin. Alias for `strip_html`.

```
my $result = nohtml $x
# or in a pipeline:
@list |> map nohtml |> p
```

noise_model_depolarize

`noise_model_depolarize` — batch 80 builtin.

```
my $result = noise_model_depolarize $x
# or in a pipeline:
@list |> map noise_model_depolarize |> p
```

nominal

`nominal` — batch 81 builtin.

```
my $result = nominal $x
# or in a pipeline:
@list |> map nominal |> p
```

none_match

`none_match` — python/ruby stdlib builtin.

```
my $result = none_match $x
# or in a pipeline:
@list |> map none_match |> p
```

nonem

`nonem` — python/ruby stdlib builtin. Alias for `none_match`.

```
my $result = nonem $x
# or in a pipeline:
@list |> map nonem |> p
```

nonempty_count

`nonempty_count` — counters (batch 4) builtin.

```
my $result = nonempty_count $x
# or in a pipeline:
@list |> map nonempty_count |> p
```

nonlinear_dual_step

`nonlinear_dual_step` — batch 41 builtin.

```
my $result = nonlinear_dual_step $x
# or in a pipeline:
@list |> map nonlinear_dual_step |> p
```

nonzero

`nonzero` — trivial numeric / predicate builtins builtin.

```
my $result = nonzero $x
# or in a pipeline:
@list |> map nonzero |> p
```

nonzero_count

`nonzero_count` — batch 72 builtin.

```
my $result = nonzero_count $x
# or in a pipeline:
@list |> map nonzero_count |> p
```

noop_val

`noop_val` — conversion / utility (batch 4) builtin.

```
my $result = noop_val $x
# or in a pipeline:
@list |> map noop_val |> p
```

nop

`nop` — conversion / utility (batch 4) builtin.

```
my $result = nop $x
# or in a pipeline:
@list |> map nop |> p
```

normal_cdf

`normal_cdf($x, $mu, $sigma)` (alias `normcdf`) — cumulative distribution function. Returns probability that a value is ≤ x.

```
p normcdf(0)              # 0.5 (half below mean)
p normcdf(1.96)           # ~0.975 (95% confidence bound)
```

normal_pdf

`normal_pdf($x, $mu, $sigma)` (alias `normpdf`) — normal/Gaussian distribution probability density function. Default: μ=0, σ=1 (standard normal).

```
p normpdf(0)              # ~0.399 (peak of standard normal)
p normpdf(0, 100, 15)     # PDF at IQ=100, mean=100, stddev=15
```

normality_chem

`normality_chem` — batch 12 builtin.

```
my $result = normality_chem $x
# or in a pipeline:
@list |> map normality_chem |> p
```

normalize_array

`normalize_array` — matrix operations (uncategorized batch) builtin.

```
my $result = normalize_array $x
# or in a pipeline:
@list |> map normalize_array |> p
```

normalize_list

`normalize_list` — functional combinators builtin.

```
my $result = normalize_list $x
# or in a pipeline:
@list |> map normalize_list |> p
```

normalize_range

`normalize_range` — matrix operations (uncategorized batch) builtin.

```
my $result = normalize_range $x
# or in a pipeline:
@list |> map normalize_range |> p
```

normalize_signal

`normalize_signal(\@signal)` (alias `normsig`) — scales signal to range [-1, 1] based on its peak absolute value. Useful for audio processing.

```
my $norm = normsig(\@audio)
p max(map { abs } @$norm)   # 1
```

normalize_spaces

`normalize_spaces` — extended stdlib builtin.

```
my $result = normalize_spaces $x
# or in a pipeline:
@list |> map normalize_spaces |> p
```

normalize_vec

`normalize_vec` — extended stdlib builtin.

```
my $result = normalize_vec $x
# or in a pipeline:
@list |> map normalize_vec |> p
```

normalize_whitespace

`normalize_whitespace` — string helpers (batch 4) builtin.

```
my $result = normalize_whitespace $x
# or in a pipeline:
@list |> map normalize_whitespace |> p
```

normarr

`normarr` — matrix operations (uncategorized batch) builtin. Alias for `normalize_array`.

```
my $result = normarr $x
# or in a pipeline:
@list |> map normarr |> p
```

not_any

`not_any` — algebraic match builtin.

```
my $result = not_any $x
# or in a pipeline:
@list |> map not_any |> p
```

not_each

`not_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = not_each $x
# or in a pipeline:
@list |> map not_each |> p
```

not_elem

`not_elem` — haskell list functions builtin.

```
my $result = not_elem $x
# or in a pipeline:
@list |> map not_elem |> p
```

not_every

`not_every` — algebraic match builtin.

```
my $result = not_every $x
# or in a pipeline:
@list |> map not_every |> p
```

notall

`notall` — list / aggregate builtin.

```
my $result = notall $x
# or in a pipeline:
@list |> map notall |> p
```

notch_filter

`notch_filter` — batch 74 builtin.

```
my $result = notch_filter $x
# or in a pipeline:
@list |> map notch_filter |> p
```

note_name_from_midi

note_name_from_midi — batch 64 builtin.

```
my $result = note_name_from_midi $x
# or in a pipeline:
@list |> map note_name_from_midi |> p
```

note_name_to_midi

note_name_to_midi — batch 30 builtin.

```
my $result = note_name_to_midi $input
```

now

now — Unix epoch seconds. Returns the current time as the number of seconds since 1970-01-01T00:00:00Z, like Perl's time(). Pass a timezone string (e.g. now("UTC") or now("America/New_York")) to get an ISO-8601 datetime string in that zone instead — that path forwards to datetime_now_tz.

```
p now              # 1778262882
p now("UTC")       # 2026-05-08T17:54:42.577935Z
my $start = now
# ... do work ...
printf "elapsed: %d s\n", now - $start
```

now_ms

now_ms — now / timestamp builtin.

```
my $result = now_ms $x
# or in a pipeline:
@list |> map now_ms |> p
```

now_ns

now_ns — now / timestamp builtin.

```
my $result = now_ns $x
# or in a pipeline:
@list |> map now_ns |> p
```

now_us

now_us — now / timestamp builtin.

```
my $result = now_us $x
# or in a pipeline:
@list |> map now_us |> p
```

nper

nper($rate, $pmt, $pv) (alias num_periods) — computes the number of periods needed to pay off a loan given rate, payment, and principal.

```
p nper(0.05/12, 500, 50000)   # ~127 months
```

npv

npv — math / numeric (uncategorized batch) builtin.

```
my $result = npv $x
# or in a pipeline:
@list |> map npv |> p
```

nrmrng

`nrmrng` — matrix operations (uncategorized batch) builtin. Alias for `normalize_range`.

```
my $result = nrmrng $x
# or in a pipeline:
@list |> map nrmrng |> p
```

nrmsp

`nrmsp` — extended stdlib builtin. Alias for `normalize_spaces`.

```
my $result = nrmsp $x
# or in a pipeline:
@list |> map nrmsp |> p
```

nrmv

`nrmv` — extended stdlib builtin. Alias for `normalize_vec`.

```
my $result = nrmv $x
# or in a pipeline:
@list |> map nrmv |> p
```

nroot

`nroot` — extended stdlib builtin. Alias for `nth_root_of`.

```
my $result = nroot $x
# or in a pipeline:
@list |> map nroot |> p
```

nrow

`nrow` — number of rows in a matrix.

```
p nrow([[1,2],[3,4],[5,6]])  # 3
```

ns_to_ms

`ns_to_ms` — file stat / path builtin.

```
my $result = ns_to_ms $input
```

ns_to_us

`ns_to_us` — file stat / path builtin.

```
my $result = ns_to_us $input
```

ntelm

`ntelm` — haskell list functions builtin. Alias for `not_elem`.

```
my $result = ntelm $x
# or in a pipeline:
@list |> map ntelm |> p
```

nth_largest

`nth_largest` — batch 9 builtin.

```
my $result = nth_largest $x
# or in a pipeline:
@list |> map nth_largest |> p
```

nth_prime

`nth_prime` — number theory / primes builtin.

```
my $result = nth_prime $x
# or in a pipeline:
@list |> map nth_prime |> p
```

nth_root

`nth_root` — math functions builtin.

```
my $result = nth_root $x
# or in a pipeline:
@list |> map nth_root |> p
```

nth_root_of

`nth_root_of` — extended stdlib builtin.

```
my $result = nth_root_of $x
# or in a pipeline:
@list |> map nth_root_of |> p
```

nth_smallest

`nth_smallest` — batch 9 builtin.

```
my $result = nth_smallest $x
# or in a pipeline:
@list |> map nth_smallest |> p
```

nth_word

`nth_word` — file stat / path builtin.

```
my $result = nth_word $x
# or in a pipeline:
@list |> map nth_word |> p
```

nthp

`nthp` — number theory / primes builtin. Alias for `nth_prime`.

```
my $result = nthp $x
# or in a pipeline:
@list |> map nthp |> p
```

nub

`nub` — haskell list functions builtin.

```
my $result = nub $x
# or in a pipeline:
@list |> map nub |> p
```

nub_by

`nub_by` — additional missing stdlib functions builtin.

```
my $result = nub_by $x
# or in a pipeline:
@list |> map nub_by |> p
```

nub_count

`nub_count` — batch 54 builtin.

```
my $result = nub_count $x
# or in a pipeline:
@list |> map nub_count |> p
```

nub_list

`nub_list` — batch 54 builtin.

```
my $result = nub_list $x
# or in a pipeline:
@list |> map nub_list |> p
```

nubb

`nubb` — additional missing stdlib functions builtin. Alias for `nub_by`.

```
my $result = nubb $x
# or in a pipeline:
@list |> map nubb |> p
```

nuclear_magneton

`nuclear_magneton` — physics constants builtin.

```
my $result = nuclear_magneton $x
# or in a pipeline:
@list |> map nuclear_magneton |> p
```

nucleation_rate_constant

`nucleation_rate_constant` — batch 38 builtin.

```
my $result = nucleation_rate_constant $x
# or in a pipeline:
@list |> map nucleation_rate_constant |> p
```

nucleolus_lp_step

`nucleolus_lp_step` — batch 43 builtin.

```
my $result = nucleolus_lp_step $x
# or in a pipeline:
@list |> map nucleolus_lp_step |> p
```

num_cpus

num_cpus — system introspection builtin.

```
my $result = num_cpus $x
# or in a pipeline:
@list |> map num_cpus |> p
```

num_divisors

num_divisors — math / numeric (uncategorized batch) builtin.

```
my $result = num_divisors $x
# or in a pipeline:
@list |> map num_divisors |> p
```

num_edges

num_edges — batch 24 builtin.

```
my $result = num_edges $x
# or in a pipeline:
@list |> map num_edges |> p
```

num_unrooted_trees

num_unrooted_trees — batch 33 builtin.

```
my $result = num_unrooted_trees $x
# or in a pipeline:
@list |> map num_unrooted_trees |> p
```

number_needed_to_treat

number_needed_to_treat — batch 61 builtin.

```
my $result = number_needed_to_treat $input
```

number_to_english

number_to_english — batch 30 builtin.

```
my $result = number_to_english $input
```

numeric_count

numeric_count — counters (batch 4) builtin.

```
my $result = numeric_count $x
# or in a pipeline:
@list |> map numeric_count |> p
```

numerical_jacobian_col

numerical_jacobian_col — batch 34 builtin.

```
my $result = numerical_jacobian_col $x
# or in a pipeline:
@list |> map numerical_jacobian_col |> p
```

nusselt_dittus_boelter

`nusselt_dittus_boelter` — batch 13 builtin.

```
my $result = nusselt_dittus_boelter $x
# or in a pipeline:
@list |> map nusselt_dittus_boelter |> p
```

nusselt_full_number

`nusselt_full_number` — batch 42 builtin.

```
my $result = nusselt_full_number $x
# or in a pipeline:
@list |> map nusselt_full_number |> p
```

nutation_iau2000a

`nutation_iau2000a` — batch 55 builtin.

```
my $result = nutation_iau2000a $x
# or in a pipeline:
@list |> map nutation_iau2000a |> p
```

nuttall_w

`nuttall_w` — batch 74 builtin.

```
my $result = nuttall_w $x
# or in a pipeline:
@list |> map nuttall_w |> p
```

nxtp

`nxtp` — number theory / primes builtin. Alias for `next_prime`.

```
my $result = nxtp $x
# or in a pipeline:
@list |> map nxtp |> p
```

nxtperm

`nxtperm` — matrix operations (uncategorized batch) builtin. Alias for `next_permutation`.

```
my $result = nxtperm $x
# or in a pipeline:
@list |> map nxtperm |> p
```

nyquist_encirclement

`nyquist_encirclement` — batch 59 builtin.

```
my $result = nyquist_encirclement $x
# or in a pipeline:
@list |> map nyquist_encirclement |> p
```

nyquist_phase

`nyquist_phase` — batch 38 builtin.

```
my $result = nyquist_phase $x
# or in a pipeline:
@list |> map nyquist_phase |> p
```

nysiis_phonetic

`nysiis_phonetic` — batch 69 builtin.

```
my $result = nysiis_phonetic $x
# or in a pipeline:
@list |> map nysiis_phonetic |> p
```

oaconvolve_step

`oaconvolve_step` — batch 74 builtin.

```
my $result = oaconvolve_step $x
# or in a pipeline:
@list |> map oaconvolve_step |> p
```

oat_hash

`oat_hash` — batch 81 builtin.

```
my $result = oat_hash $x
# or in a pipeline:
@list |> map oat_hash |> p
```

object_encoding

`object_encoding` — batch 71 builtin.

```
my $result = object_encoding $x
# or in a pipeline:
@list |> map object_encoding |> p
```

object_entries

`object_entries` — javascript array/object methods builtin.

```
my $result = object_entries $x
# or in a pipeline:
@list |> map object_entries |> p
```

object_from_entries

`object_from_entries` — javascript array/object methods builtin.

```
my $result = object_from_entries $x
# or in a pipeline:
@list |> map object_from_entries |> p
```

object_keys

`object_keys` — javascript array/object methods builtin.

```
my $result = object_keys $x
# or in a pipeline:
@list |> map object_keys |> p
```

object_values

`object_values` — javascript array/object methods builtin.

```
my $result = object_values $x
# or in a pipeline:
@list |> map object_values |> p
```

oct_of

`oct_of` — base conversion builtin. Alias for `to_oct`.

```
my $result = oct_of $x
# or in a pipeline:
@list |> map oct_of |> p
```

octahedral_number

`octahedral_number` — batch 10 builtin.

```
my $result = octahedral_number $x
# or in a pipeline:
@list |> map octahedral_number |> p
```

ocv_lithium_ion_step

`ocv_lithium_ion_step` — batch 38 builtin.

```
my $result = ocv_lithium_ion_step $x
# or in a pipeline:
@list |> map ocv_lithium_ion_step |> p
```

odd

`odd` — trivial numeric / predicate builtins builtin.

```
my $result = odd $x
# or in a pipeline:
@list |> map odd |> p
```

odddyield

`odddyield` — batch 81 builtin.

```
my $result = odddyield $x
# or in a pipeline:
@list |> map odddyield |> p
```

odds_ratio_2x2

`odds_ratio_2x2` — batch 61 builtin.

```
my $result = odds_ratio_2x2 $x
# or in a pipeline:
@list |> map odds_ratio_2x2 |> p
```

ode45_step

`ode45_step` — batch 72 builtin.

```
my $result = ode45_step $x
# or in a pipeline:
@list |> map ode45_step |> p
```

ode_lsoda

`ode_lsoda` — batch 72 builtin.

```
my $result = ode_lsoda $x
# or in a pipeline:
@list |> map ode_lsoda |> p
```

oents

`oents` — javascript array/object methods builtin. Alias for `object_entries`.

```
my $result = oents $x
# or in a pipeline:
@list |> map oents |> p
```

ofents

`ofents` — javascript array/object methods builtin. Alias for `object_from_entries`.

```
my $result = ofents $x
# or in a pipeline:
@list |> map ofents |> p
```

off

`off` — color / ansi builtin. Alias for `red`.

```
my $result = off $x
# or in a pipeline:
@list |> map off |> p
```

offset

`offset` — batch 57 builtin.

```
my $result = offset $x
# or in a pipeline:
@list |> map offset |> p
```

offset_each

`offset_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = offset_each $x
# or in a pipeline:
@list |> map offset_each |> p
```

ohm_voltage

`ohm_voltage` — batch 23 builtin.

```
my $result = ohm_voltage $x
# or in a pipeline:
@list |> map ohm_voltage |> p
```

ohms_law_i

`ohms_law_i` — physics formulas builtin.

```
my $result = ohms_law_i $x
# or in a pipeline:
@list |> map ohms_law_i |> p
```

ohms_law_r

`ohms_law_r` — physics formulas builtin.

```
my $result = ohms_law_r $x
# or in a pipeline:
@list |> map ohms_law_r |> p
```

ohms_law_v

`ohms_law_v` — physics formulas builtin.

```
my $result = ohms_law_v $x
# or in a pipeline:
@list |> map ohms_law_v |> p
```

okeys

`okeys` — javascript array/object methods builtin. Alias for `object_keys`.

```
my $result = okeys $x
# or in a pipeline:
@list |> map okeys |> p
```

oklab_to_rgb

`oklab_to_rgb` — batch 82 builtin.

```
my $result = oklab_to_rgb $input
```

ols_adjusted_r2

`ols_adjusted_r2` — batch 36 builtin.

```
my $result = ols_adjusted_r2 $x
# or in a pipeline:
@list |> map ols_adjusted_r2 |> p
```

ols_estimator

`ols_estimator` — batch 36 builtin.

```
my $result = ols_estimator $x
# or in a pipeline:
@list |> map ols_estimator |> p
```

ols_r_squared

`ols_r_squared` — batch 36 builtin.

```
my $result = ols_r_squared $x
# or in a pipeline:
@list |> map ols_r_squared |> p
```

ols_residual_variance

`ols_residual_variance` — batch 36 builtin.

```
my $result = ols_residual_variance $x
# or in a pipeline:
@list |> map ols_residual_variance |> p
```

omega_equation_step

`omega_equation_step` — batch 42 builtin.

```
my $result = omega_equation_step $x
# or in a pipeline:
@list |> map omega_equation_step |> p
```

omega_m_at_z

`omega_m_at_z` — batch 31 builtin.

```
my $result = omega_m_at_z $x
# or in a pipeline:
@list |> map omega_m_at_z |> p
```

omega_ratio

`omega_ratio` — batches 17-18 builtin.

```
my $result = omega_ratio $x
# or in a pipeline:
@list |> map omega_ratio |> p
```

omit

`omit` — hash ops (batch 2) builtin. Alias for `omit_keys`.

```
my $result = omit $x
# or in a pipeline:
@list |> map omit |> p
```

omit_keys

`omit_keys` — hash ops (batch 2) builtin.

```
my $result = omit_keys $x
# or in a pipeline:
@list |> map omit_keys |> p
```

omori_aftershock

`omori_aftershock` — batch 65 builtin.

```
my $result = omori_aftershock $x
# or in a pipeline:
@list |> map omori_aftershock |> p
```

one

`one` — python/ruby stdlib builtin.

```
my $result = one $x
# or in a pipeline:
@list |> map one |> p
```

one_hot

one_hot — batch 27 builtin.

```
my $result = one_hot $x
# or in a pipeline:
@list |> map one_hot |> p
```

ones

ones — matrix / linear algebra builtin.

```
my $result = ones $x
# or in a pipeline:
@list |> map ones |> p
```

ones_mat

ones_mat — matrix / linear algebra builtin. Alias for ones_matrix.

```
my $result = ones_mat $x
# or in a pipeline:
@list |> map ones_mat |> p
```

ones_matrix

ones_matrix — matrix / linear algebra builtin.

```
my $result = ones_matrix $x
# or in a pipeline:
@list |> map ones_matrix |> p
```

only_alnum

only_alnum — string helpers (batch 4) builtin.

```
my $result = only_alnum $x
# or in a pipeline:
@list |> map only_alnum |> p
```

only_alpha

only_alpha — string helpers (batch 4) builtin.

```
my $result = only_alpha $x
# or in a pipeline:
@list |> map only_alpha |> p
```

only_ascii

only_ascii — string helpers (batch 4) builtin.

```
my $result = only_ascii $x
# or in a pipeline:
@list |> map only_ascii |> p
```

only_digits

only_digits — string helpers (batch 4) builtin.

```
my $result = only_digits $x
# or in a pipeline:
@list |> map only_digits |> p
```

onsager_relation_two_species

onsager_relation_two_species — batch 38 builtin.

```
my $result = onsager_relation_two_species $x
# or in a pipeline:
@list |> map onsager_relation_two_species |> p
```

open_pipe_harmonic

open_pipe_harmonic — batch 23 builtin.

```
my $result = open_pipe_harmonic $x
# or in a pipeline:
@list |> map open_pipe_harmonic |> p
```

operad_count_two

operad_count_two — batch 37 builtin.

```
my $result = operad_count_two $x
# or in a pipeline:
@list |> map operad_count_two |> p
```

oppenheimer_volkoff_step

oppenheimer_volkoff_step — batch 39 builtin.

```
my $result = oppenheimer_volkoff_step $x
# or in a pipeline:
@list |> map oppenheimer_volkoff_step |> p
```

ops_plus

ops_plus — batch 56 builtin.

```
my $result = ops_plus $x
# or in a pipeline:
@list |> map ops_plus |> p
```

optical_flow_farneback

optical_flow_farneback — batch 77 builtin.

```
my $result = optical_flow_farneback $x
# or in a pipeline:
@list |> map optical_flow_farneback |> p
```

optical_flow_lk

optical_flow_lk — batch 77 builtin.

```
my $result = optical_flow_lk $x
# or in a pipeline:
@list |> map optical_flow_lk |> p
```

optical_flow_lk_step

optical_flow_lk_step — batch 40 builtin.

```
my $result = optical_flow_lk_step $x
# or in a pipeline:
@list |> map optical_flow_lk_step |> p
```

optical_scalars_step

optical_scalars_step — batch 39 builtin.

```
my $result = optical_scalars_step $x
# or in a pipeline:
@list |> map optical_scalars_step |> p
```

or

or — control flow builtin.

```
my $result = or $x
# or in a pipeline:
@list |> map or |> p
```

or_list

or_list — additional missing stdlib functions builtin.

```
my $result = or_list $x
# or in a pipeline:
@list |> map or_list |> p
```

or_opt_delta

or_opt_delta — batch 41 builtin.

```
my $result = or_opt_delta $x
# or in a pipeline:
@list |> map or_opt_delta |> p
```

orb_keypoints

orb_keypoints — batch 77 builtin.

```
my $result = orb_keypoints $x
# or in a pipeline:
@list |> map orb_keypoints |> p
```

orbit_eccentric_anomaly

orbit_eccentric_anomaly — batch 55 builtin.

```
my $result = orbit_eccentric_anomaly $x
# or in a pipeline:
@list |> map orbit_eccentric_anomaly |> p
```

orbit_kepler3

`orbit_kepler3` — batch 55 builtin.

```
my $result = orbit_kepler3 $x
# or in a pipeline:
@list |> map orbit_kepler3 |> p
```

orbital_eccentricity

`orbital_eccentricity` — batches 14-16 builtin.

```
my $result = orbital_eccentricity $x
# or in a pipeline:
@list |> map orbital_eccentricity |> p
```

orbital_period

`orbital_period($mass, $radius)` (alias `orbper`) — period for circular orbit: T = 2π√(r³/GM). Returns seconds.

```
my $t = orbper       # ~5560 s (ISS orbital period)
p $t / 60            # ~93 minutes
```

orbital_period_au

`orbital_period_au` — batch 55 builtin.

```
my $result = orbital_period_au $x
# or in a pipeline:
@list |> map orbital_period_au |> p
```

orbital_velocity

`orbital_velocity($mass, $radius)` (alias `orbvel`) — velocity for circular orbit: v = √(GM/r). Defaults to LEO around Earth.

```
p orbvel    # ~7672 m/s (ISS orbital speed)
```

ordered_probit_threshold

`ordered_probit_threshold` — batch 36 builtin.

```
my $result = ordered_probit_threshold $x
# or in a pipeline:
@list |> map ordered_probit_threshold |> p
```

ordered_set

`ordered_set` — data structure helpers builtin.

```
my $result = ordered_set $x
# or in a pipeline:
@list |> map ordered_set |> p
```

ordinal_date

`ordinal_date` — batch 10 builtin. Alias for `iso_ordinal_date`.

```
my $result = ordinal_date $x
# or in a pipeline:
@list |> map ordinal_date |> p
```

ordinalize

`ordinalize` — extended stdlib builtin.

```
my $result = ordinalize $x
# or in a pipeline:
@list |> map ordinalize |> p
```

ordn

`ordn` — extended stdlib builtin. Alias for `ordinalize`.

```
my $result = ordn $x
# or in a pipeline:
@list |> map ordn |> p
```

orifice_velocity

`orifice_velocity` — batch 13 builtin.

```
my $result = orifice_velocity $x
# or in a pipeline:
@list |> map orifice_velocity |> p
```

orl

`orl` — additional missing stdlib functions builtin. Alias for `or_list`.

```
my $result = orl $x
# or in a pipeline:
@list |> map orl |> p
```

orthogonalize_vectors

`orthogonalize_vectors` — batch 11 builtin.

```
my $result = orthogonalize_vectors $x
# or in a pipeline:
@list |> map orthogonalize_vectors |> p
```

os_acpi_state_transition

`os_acpi_state_transition` — batch 50 builtin.

```
my $result = os_acpi_state_transition $x
# or in a pipeline:
@list |> map os_acpi_state_transition |> p
```

os_amd_pstate_target

`os_amd_pstate_target` — batch 50 builtin.

```
my $result = os_amd_pstate_target $x
# or in a pipeline:
@list |> map os_amd_pstate_target |> p
```

os_anticipation_window

`os_anticipation_window` — batch 50 builtin.

```
my $result = os_anticipation_window $x
# or in a pipeline:
@list |> map os_anticipation_window |> p
```

os_apic_irq_priority

`os_apic_irq_priority` — batch 50 builtin.

```
my $result = os_apic_irq_priority $x
# or in a pipeline:
@list |> map os_apic_irq_priority |> p
```

os_apparmor_profile_active

`os_apparmor_profile_active` — batch 50 builtin.

```
my $result = os_apparmor_profile_active $x
# or in a pipeline:
@list |> map os_apparmor_profile_active |> p
```

os_arch

`os_arch` — system introspection builtin.

```
my $result = os_arch $x
# or in a pipeline:
@list |> map os_arch |> p
```

os_audit_event_priority

`os_audit_event_priority` — batch 50 builtin.

```
my $result = os_audit_event_priority $x
# or in a pipeline:
@list |> map os_audit_event_priority |> p
```

os_battery_capacity_pct

`os_battery_capacity_pct` — batch 50 builtin.

```
my $result = os_battery_capacity_pct $x
# or in a pipeline:
@list |> map os_battery_capacity_pct |> p
```

os_buddy_order_pick

`os_buddy_order_pick` — batch 50 builtin.

```
my $result = os_buddy_order_pick $x
# or in a pipeline:
@list |> map os_buddy_order_pick |> p
```

os_buffer_cache_pages

`os_buffer_cache_pages` — batch 50 builtin.

```
my $result = os_buffer_cache_pages $x
# or in a pipeline:
@list |> map os_buffer_cache_pages |> p
```

os_c_state_residency

`os_c_state_residency` — batch 50 builtin.

```
my $result = os_c_state_residency $x
# or in a pipeline:
@list |> map os_c_state_residency |> p
```

os_capability_check

`os_capability_check` — batch 50 builtin.

```
my $result = os_capability_check $x
# or in a pipeline:
@list |> map os_capability_check |> p
```

os_cfs_load_balance_step

`os_cfs_load_balance_step` — batch 50 builtin.

```
my $result = os_cfs_load_balance_step $x
# or in a pipeline:
@list |> map os_cfs_load_balance_step |> p
```

os_cgroup_v1_count

`os_cgroup_v1_count` — batch 50 builtin.

```
my $result = os_cgroup_v1_count $x
# or in a pipeline:
@list |> map os_cgroup_v1_count |> p
```

os_cgroup_v2_count

`os_cgroup_v2_count` — batch 50 builtin.

```
my $result = os_cgroup_v2_count $x
# or in a pipeline:
@list |> map os_cgroup_v2_count |> p
```

os_compact_memory_step

`os_compact_memory_step` — batch 50 builtin.

```
my $result = os_compact_memory_step $x
# or in a pipeline:
@list |> map os_compact_memory_step |> p
```

os_completely_fair_vruntime

`os_completely_fair_vruntime` — batch 50 builtin.

```
my $result = os_completely_fair_vruntime $x
# or in a pipeline:
@list |> map os_completely_fair_vruntime |> p
```

os_copy_on_write_check

`os_copy_on_write_check` — batch 50 builtin.

```
my $result = os_copy_on_write_check $x
# or in a pipeline:
@list |> map os_copy_on_write_check |> p
```

os_cpu_affinity_score

os_cpu_affinity_score — batch 50 builtin.

```
my $result = os_cpu_affinity_score $x
# or in a pipeline:
@list |> map os_cpu_affinity_score |> p
```

os_cpufreq_governor_step

os_cpufreq_governor_step — batch 50 builtin.

```
my $result = os_cpufreq_governor_step $x
# or in a pipeline:
@list |> map os_cpufreq_governor_step |> p
```

os_demand_paging_step

os_demand_paging_step — batch 50 builtin.

```
my $result = os_demand_paging_step $x
# or in a pipeline:
@list |> map os_demand_paging_step |> p
```

os_dirty_page_threshold

os_dirty_page_threshold — batch 50 builtin.

```
my $result = os_dirty_page_threshold $x
# or in a pipeline:
@list |> map os_dirty_page_threshold |> p
```

os_disk_rotational_lat

os_disk_rotational_lat — batch 50 builtin.

```
my $result = os_disk_rotational_lat $x
# or in a pipeline:
@list |> map os_disk_rotational_lat |> p
```

os_disk_seek_time

os_disk_seek_time — batch 50 builtin.

```
my $result = os_disk_seek_time $x
# or in a pipeline:
@list |> map os_disk_seek_time |> p
```

os_disk_transfer_time

os_disk_transfer_time — batch 50 builtin.

```
my $result = os_disk_transfer_time $x
# or in a pipeline:
@list |> map os_disk_transfer_time |> p
```

os_dmesg_severity_level

os_dmesg_severity_level — batch 50 builtin.

```
my $result = os_dmesg_severity_level $x
# or in a pipeline:
@list |> map os_dmesg_severity_level |> p
```

os_dvfs_step

os_dvfs_step — batch 50 builtin.

```
my $result = os_dvfs_step $x
# or in a pipeline:
@list |> map os_dvfs_step |> p
```

os_eas_energy_estimate

os_eas_energy_estimate — batch 50 builtin.

```
my $result = os_eas_energy_estimate $x
# or in a pipeline:
@list |> map os_eas_energy_estimate |> p
```

os_eevdf_eligible

os_eevdf_eligible — batch 50 builtin.

```
my $result = os_eevdf_eligible $x
# or in a pipeline:
@list |> map os_eevdf_eligible |> p
```

os_elevator_step

os_elevator_step — batch 50 builtin.

```
my $result = os_elevator_step $x
# or in a pipeline:
@list |> map os_elevator_step |> p
```

os_epoll_ctl_count

os_epoll_ctl_count — batch 50 builtin.

```
my $result = os_epoll_ctl_count $x
# or in a pipeline:
@list |> map os_epoll_ctl_count |> p
```

os_family

os_family — OS family string: "unix" or "windows". From std::env::consts::FAMILY.

```
p os_family                             # unix
die "unix only" unless os_family eq "unix"
```

os_file_max_value

os_file_max_value — batch 50 builtin.

```
my $result = os_file_max_value $x
# or in a pipeline:
@list |> map os_file_max_value |> p
```

os_frequency_scaling_step

os_frequency_scaling_step — batch 50 builtin.

```
my $result = os_frequency_scaling_step $x
# or in a pipeline:
@list |> map os_frequency_scaling_step |> p
```

os_huge_page_threshold

`os_huge_page_threshold` — batch 50 builtin.

```
my $result = os_huge_page_threshold $x
# or in a pipeline:
@list |> map os_huge_page_threshold |> p
```

os_idle_state_select

`os_idle_state_select` — batch 50 builtin.

```
my $result = os_idle_state_select $x
# or in a pipeline:
@list |> map os_idle_state_select |> p
```

os_inotify_event_count

`os_inotify_event_count` — batch 50 builtin.

```
my $result = os_inotify_event_count $x
# or in a pipeline:
@list |> map os_inotify_event_count |> p
```

os_inotify_max_watches

`os_inotify_max_watches` — batch 50 builtin.

```
my $result = os_inotify_max_watches $x
# or in a pipeline:
@list |> map os_inotify_max_watches |> p
```

os_intel_pstate_target

`os_intel_pstate_target` — batch 50 builtin.

```
my $result = os_intel_pstate_target $x
# or in a pipeline:
@list |> map os_intel_pstate_target |> p
```

os_io_scheduler_bfq_step

`os_io_scheduler_bfq_step` — batch 50 builtin.

```
my $result = os_io_scheduler_bfq_step $x
# or in a pipeline:
@list |> map os_io_scheduler_bfq_step |> p
```

os_io_scheduler_cfq_step

`os_io_scheduler_cfq_step` — batch 50 builtin.

```
my $result = os_io_scheduler_cfq_step $x
# or in a pipeline:
@list |> map os_io_scheduler_cfq_step |> p
```

os_io_scheduler_deadline

`os_io_scheduler_deadline` — batch 50 builtin.

```
my $result = os_io_scheduler_deadline $x
# or in a pipeline:
@list |> map os_io_scheduler_deadline |> p
```

os_io_scheduler_kyber_step

os_io_scheduler_kyber_step — batch 50 builtin.

```
my $result = os_io_scheduler_kyber_step $x
# or in a pipeline:
@list |> map os_io_scheduler_kyber_step |> p
```

os_io_scheduler_mq_deadline

os_io_scheduler_mq_deadline — batch 50 builtin.

```
my $result = os_io_scheduler_mq_deadline $x
# or in a pipeline:
@list |> map os_io_scheduler_mq_deadline |> p
```

os_io_scheduler_noop_step

os_io_scheduler_noop_step — batch 50 builtin.

```
my $result = os_io_scheduler_noop_step $x
# or in a pipeline:
@list |> map os_io_scheduler_noop_step |> p
```

os_io_uring_cqe_count

os_io_uring_cqe_count — batch 50 builtin.

```
my $result = os_io_uring_cqe_count $x
# or in a pipeline:
@list |> map os_io_uring_cqe_count |> p
```

os_io_uring_sqe_count

os_io_uring_sqe_count — batch 50 builtin.

```
my $result = os_io_uring_sqe_count $x
# or in a pipeline:
@list |> map os_io_uring_sqe_count |> p
```

os_iommu_domain_step

os_iommu_domain_step — batch 50 builtin.

```
my $result = os_iommu_domain_step $x
# or in a pipeline:
@list |> map os_iommu_domain_step |> p
```

os_kasan_shadow_offset

os_kasan_shadow_offset — batch 50 builtin.

```
my $result = os_kasan_shadow_offset $x
# or in a pipeline:
@list |> map os_kasan_shadow_offset |> p
```

os_kfence_alloc_index

os_kfence_alloc_index — batch 50 builtin.

```
my $result = os_kfence_alloc_index $x
# or in a pipeline:
@list |> map os_kfence_alloc_index |> p
```

os_kfence_check

`os_kfence_check` — batch 50 builtin.

```
my $result = os_kfence_check $x
# or in a pipeline:
@list |> map os_kfence_check |> p
```

os_kqueue_event_count

`os_kqueue_event_count` — batch 50 builtin.

```
my $result = os_kqueue_event_count $x
# or in a pipeline:
@list |> map os_kqueue_event_count |> p
```

os_kswapd_wake_threshold

`os_kswapd_wake_threshold` — batch 50 builtin.

```
my $result = os_kswapd_wake_threshold $x
# or in a pipeline:
@list |> map os_kswapd_wake_threshold |> p
```

os_kvm_vmcs_field_offset

`os_kvm_vmcs_field_offset` — batch 50 builtin.

```
my $result = os_kvm_vmcs_field_offset $x
# or in a pipeline:
@list |> map os_kvm_vmcs_field_offset |> p
```

os_load_average_decay

`os_load_average_decay` — batch 50 builtin.

```
my $result = os_load_average_decay $x
# or in a pipeline:
@list |> map os_load_average_decay |> p
```

os_load_iowait_avg

`os_load_iowait_avg` — batch 50 builtin.

```
my $result = os_load_iowait_avg $x
# or in a pipeline:
@list |> map os_load_iowait_avg |> p
```

os_load_proc_avg

`os_load_proc_avg` — batch 50 builtin.

```
my $result = os_load_proc_avg $x
# or in a pipeline:
@list |> map os_load_proc_avg |> p
```

os_load_user_avg

`os_load_user_avg` — batch 50 builtin.

```
my $result = os_load_user_avg $x
# or in a pipeline:
@list |> map os_load_user_avg |> p
```

os_lottery_ticket_count

os_lottery_ticket_count — batch 50 builtin.

```
my $result = os_lottery_ticket_count $x
# or in a pipeline:
@list |> map os_lottery_ticket_count |> p
```

os_mlfq_demote_step

os_mlfq_demote_step — batch 50 builtin.

```
my $result = os_mlfq_demote_step $x
# or in a pipeline:
@list |> map os_mlfq_demote_step |> p
```

os_mlfq_promote_step

os_mlfq_promote_step — batch 50 builtin.

```
my $result = os_mlfq_promote_step $x
# or in a pipeline:
@list |> map os_mlfq_promote_step |> p
```

os_msi_x_vector_count

os_msi_x_vector_count — batch 50 builtin.

```
my $result = os_msi_x_vector_count $x
# or in a pipeline:
@list |> map os_msi_x_vector_count |> p
```

os_name

os_name — system introspection builtin.

```
my $result = os_name $x
# or in a pipeline:
@list |> map os_name |> p
```

os_namespace_isolation

os_namespace_isolation — batch 50 builtin.

```
my $result = os_namespace_isolation $x
# or in a pipeline:
@list |> map os_namespace_isolation |> p
```

os_numa_node_distance

os_numa_node_distance — batch 50 builtin.

```
my $result = os_numa_node_distance $x
# or in a pipeline:
@list |> map os_numa_node_distance |> p
```

os_oom_kill_score

os_oom_kill_score — batch 50 builtin.

```
my $result = os_oom_kill_score $x
# or in a pipeline:
@list |> map os_oom_kill_score |> p
```

os_oom_score_step

`os_oom_score_step` — batch 50 builtin.

```
my $result = os_oom_score_step $x
# or in a pipeline:
@list |> map os_oom_score_step |> p
```

os_open_files_count

`os_open_files_count` — batch 50 builtin.

```
my $result = os_open_files_count $x
# or in a pipeline:
@list |> map os_open_files_count |> p
```

os_p_state_voltage

`os_p_state_voltage` — batch 50 builtin.

```
my $result = os_p_state_voltage $x
# or in a pipeline:
@list |> map os_p_state_voltage |> p
```

os_page_replacement_2q

`os_page_replacement_2q` — batch 50 builtin.

```
my $result = os_page_replacement_2q $x
# or in a pipeline:
@list |> map os_page_replacement_2q |> p
```

os_page_replacement_clock

`os_page_replacement_clock` — batch 50 builtin.

```
my $result = os_page_replacement_clock $x
# or in a pipeline:
@list |> map os_page_replacement_clock |> p
```

os_page_replacement_lru

`os_page_replacement_lru` — batch 50 builtin.

```
my $result = os_page_replacement_lru $x
# or in a pipeline:
@list |> map os_page_replacement_lru |> p
```

os_pci_bus_address

`os_pci_bus_address` — batch 50 builtin.

```
my $result = os_pci_bus_address $x
# or in a pipeline:
@list |> map os_pci_bus_address |> p
```

os_per_cpu_cache_size

os_per_cpu_cache_size — batch 50 builtin.

```
my $result = os_per_cpu_cache_size $x
# or in a pipeline:
@list |> map os_per_cpu_cache_size |> p
```

os_pid_max_value

os_pid_max_value — batch 50 builtin.

```
my $result = os_pid_max_value $x
# or in a pipeline:
@list |> map os_pid_max_value |> p
```

os_powertop_score

os_powertop_score — batch 50 builtin.

```
my $result = os_powertop_score $x
# or in a pipeline:
@list |> map os_powertop_score |> p
```

os_pre_fetch_window

os_pre_fetch_window — batch 50 builtin.

```
my $result = os_pre_fetch_window $x
# or in a pipeline:
@list |> map os_pre_fetch_window |> p
```

os_pressure_stall_step

os_pressure_stall_step — batch 50 builtin.

```
my $result = os_pressure_stall_step $x
# or in a pipeline:
@list |> map os_pressure_stall_step |> p
```

os_priority_aging_step

os_priority_aging_step — batch 50 builtin.

```
my $result = os_priority_aging_step $x
# or in a pipeline:
@list |> map os_priority_aging_step |> p
```

os_psi_avg10_step

os_psi_avg10_step — batch 50 builtin.

```
my $result = os_psi_avg10_step $x
# or in a pipeline:
@list |> map os_psi_avg10_step |> p
```

os_psi_avg300_step

os_psi_avg300_step — batch 50 builtin.

```
my $result = os_psi_avg300_step $x
# or in a pipeline:
@list |> map os_psi_avg300_step |> p
```

os_psi_avg60_step

os_psi_avg60_step — batch 50 builtin.

```
my $result = os_psi_avg60_step $x
# or in a pipeline:
@list |> map os_psi_avg60_step |> p
```

os_round_robin_quantum

os_round_robin_quantum — batch 50 builtin.

```
my $result = os_round_robin_quantum $x
# or in a pipeline:
@list |> map os_round_robin_quantum |> p
```

os_runqueue_depth

os_runqueue_depth — batch 50 builtin.

```
my $result = os_runqueue_depth $x
# or in a pipeline:
@list |> map os_runqueue_depth |> p
```

os_seccomp_filter_step

os_seccomp_filter_step — batch 50 builtin.

```
my $result = os_seccomp_filter_step $x
# or in a pipeline:
@list |> map os_seccomp_filter_step |> p
```

os_selinux_context_match

os_selinux_context_match — batch 50 builtin.

```
my $result = os_selinux_context_match $x
# or in a pipeline:
@list |> map os_selinux_context_match |> p
```

os_slab_color_offset

os_slab_color_offset — batch 50 builtin.

```
my $result = os_slab_color_offset $x
# or in a pipeline:
@list |> map os_slab_color_offset |> p
```

os_slub_object_size_round

os_slub_object_size_round — batch 50 builtin.

```
my $result = os_slub_object_size_round $x
# or in a pipeline:
@list |> map os_slub_object_size_round |> p
```

os_smack_label_compare

os_smack_label_compare — batch 50 builtin.

```
my $result = os_smack_label_compare $x
# or in a pipeline:
@list |> map os_smack_label_compare |> p
```

os_smt_threading_share

os_smt_threading_share — batch 50 builtin.

```
my $result = os_smt_threading_share $x
# or in a pipeline:
@list |> map os_smt_threading_share |> p
```

os_socket_max_value

os_socket_max_value — batch 50 builtin.

```
my $result = os_socket_max_value $x
# or in a pipeline:
@list |> map os_socket_max_value |> p
```

os_stride_pass_step

os_stride_pass_step — batch 50 builtin.

```
my $result = os_stride_pass_step $x
# or in a pipeline:
@list |> map os_stride_pass_step |> p
```

os_swap_pressure_score

os_swap_pressure_score — batch 50 builtin.

```
my $result = os_swap_pressure_score $x
# or in a pipeline:
@list |> map os_swap_pressure_score |> p
```

os_swappiness_factor

os_swappiness_factor — batch 50 builtin.

```
my $result = os_swappiness_factor $x
# or in a pipeline:
@list |> map os_swappiness_factor |> p
```

os_systemd_journal_size

os_systemd_journal_size — batch 50 builtin.

```
my $result = os_systemd_journal_size $x
# or in a pipeline:
@list |> map os_systemd_journal_size |> p
```

os_thermal_zone_trip

os_thermal_zone_trip — batch 50 builtin.

```
my $result = os_thermal_zone_trip $x
# or in a pipeline:
@list |> map os_thermal_zone_trip |> p
```

os_thrashing_threshold

os_thrashing_threshold — batch 50 builtin.

```
my $result = os_thrashing_threshold $x
# or in a pipeline:
@list |> map os_thrashing_threshold |> p
```

os_thread_max_value

os_thread_max_value — batch 50 builtin.

```
my $result = os_thread_max_value $x
# or in a pipeline:
@list |> map os_thread_max_value |> p
```

os_thread_migration_cost

os_thread_migration_cost — batch 50 builtin.

```
my $result = os_thread_migration_cost $x
# or in a pipeline:
@list |> map os_thread_migration_cost |> p
```

os_throttle_temperature

os_throttle_temperature — batch 50 builtin.

```
my $result = os_throttle_temperature $x
# or in a pipeline:
@list |> map os_throttle_temperature |> p
```

os_transparent_hugepage

os_transparent_hugepage — batch 50 builtin.

```
my $result = os_transparent_hugepage $x
# or in a pipeline:
@list |> map os_transparent_hugepage |> p
```

os_version

os_version — OS kernel version/release string from uname. Returns undef on non-unix.

```
p os_version                            # 25.4.0
p "kernel: " . os_version
```

os_voltage_scaling_step

os_voltage_scaling_step — batch 50 builtin.

```
my $result = os_voltage_scaling_step $x
# or in a pipeline:
@list |> map os_voltage_scaling_step |> p
```

os_working_set_size

os_working_set_size — batch 50 builtin.

```
my $result = os_working_set_size $x
# or in a pipeline:
@list |> map os_working_set_size |> p
```

os_writeback_step

os_writeback_step — batch 50 builtin.

```
my $result = os_writeback_step $x
# or in a pipeline:
@list |> map os_writeback_step |> p
```

os_zero_page_optimization

`os_zero_page_optimization` — batch 50 builtin.

```
my $result = os_zero_page_optimization $x
# or in a pipeline:
@list |> map os_zero_page_optimization |> p
```

os_zram_compress_ratio

`os_zram_compress_ratio` — batch 50 builtin.

```
my $result = os_zram_compress_ratio $x
# or in a pipeline:
@list |> map os_zram_compress_ratio |> p
```

os_zswap_compress_ratio

`os_zswap_compress_ratio` — batch 50 builtin.

```
my $result = os_zswap_compress_ratio $x
# or in a pipeline:
@list |> map os_zswap_compress_ratio |> p
```

osmotic_coefficient_pitzer

`osmotic_coefficient_pitzer` — batch 38 builtin.

```
my $result = osmotic_coefficient_pitzer $x
# or in a pipeline:
@list |> map osmotic_coefficient_pitzer |> p
```

osmotic_pressure

`osmotic_pressure` — batch 21 builtin.

```
my $result = osmotic_pressure $x
# or in a pipeline:
@list |> map osmotic_pressure |> p
```

otto_efficiency

`otto_efficiency` — batch 12 builtin.

```
my $result = otto_efficiency $input
```

oursync

Package-global counterpart of `mysync`. Same `Arc<Mutex>` backing, but the binding is keyed by the package-qualified stash name (`Pkg::x`) so all packages and parallel workers share one cell.

```
package Counter
oursync $total = 0
fn bump { $total++ }
package main
fan 10000 { Counter::bump() }
p $Counter::total   # 10000
```

Use `oursync` for cross-package shared state; use `mysync` for lexically scoped shared state.

out_degree_directed

`out_degree_directed` — batch 13 builtin.

```
my $result = out_degree_directed $x
# or in a pipeline:
@list |> map out_degree_directed |> p
```

outer

`outer` — outer product of two vectors. Returns matrix where M[i][j] = v1[i] * v2[j].

```
my $m = outer([1,2,3], [10,20])  # [[10,20],[20,40],[30,60]]
```

outlier_iqr_q

`outlier_iqr_q` — batch 11 builtin.

```
my $result = outlier_iqr_q $x
# or in a pipeline:
@list |> map outlier_iqr_q |> p
```

outliers

`outliers` — statistics (extended) builtin. Alias for `outliers_iqr`.

```
my $result = outliers $x
# or in a pipeline:
@list |> map outliers |> p
```

outliers_iqr

`outliers_iqr(@data)` — identifies outliers using the IQR method (1.5 × IQR beyond Q1/Q3). Returns an arrayref of outlier values. Standard approach for detecting unusual data points.

```
my @data = (1, 2, 3, 4, 5, 100)
my $out = outliers_iqr(@data)
p @$out   # (100)
```

ovals

`ovals` — javascript array/object methods builtin. Alias for `object_values`.

```
my $result = ovals $x
# or in a pipeline:
@list |> map ovals |> p
```

overall_efficiency_pec

`overall_efficiency_pec` — batch 38 builtin.

```
my $result = overall_efficiency_pec $x
# or in a pipeline:
@list |> map overall_efficiency_pec |> p
```

overlap_coefficient

`overlap_coefficient` — extended stdlib batch 3 builtin.

```
my $result = overlap_coefficient $x
# or in a pipeline:
@list |> map overlap_coefficient |> p
```

overlapcoef

`overlapcoef` — extended stdlib batch 3 builtin. Alias for `overlap_coefficient`.

```
my $result = overlapcoef $x
# or in a pipeline:
@list |> map overlapcoef |> p
```

overline

`overline` — color / ansi builtin. Alias for `red`.

```
my $result = overline $x
# or in a pipeline:
@list |> map overline |> p
```

overpotential_her

`overpotential_her` — batch 38 builtin.

```
my $result = overpotential_her $x
# or in a pipeline:
@list |> map overpotential_her |> p
```

overpotential_oer

`overpotential_oer` — batch 38 builtin.

```
my $result = overpotential_oer $x
# or in a pipeline:
@list |> map overpotential_oer |> p
```

owen_t

`owen_t` — batch 29 builtin.

```
my $result = owen_t $x
# or in a pipeline:
@list |> map owen_t |> p
```

owens_t

`owens_t` — batch 72 builtin.

```
my $result = owens_t $x
# or in a pipeline:
@list |> map owens_t |> p
```

oxidation_number

`oxidation_number` — batch 63 builtin.

```
my $result = oxidation_number $x
# or in a pipeline:
@list |> map oxidation_number |> p
```

oz_to_g

`oz_to_g` — unit conversions builtin.

```
my $result = oz_to_g $input
```

ozone_dobson_total

`ozone_dobson_total` — batch 42 builtin.

```
my $result = ozone_dobson_total $x
# or in a pipeline:
@list |> map ozone_dobson_total |> p
```

ozsvath_szabo_tau

`ozsvath_szabo_tau` — batch 37 builtin.

```
my $result = ozsvath_szabo_tau $x
# or in a pipeline:
@list |> map ozsvath_szabo_tau |> p
```

pa_to_atm

`pa_to_atm` — batch 30 builtin.

```
my $result = pa_to_atm $input
```

pacf

`pacf_fn` (alias `pacf`) — partial autocorrelation function via Durbin-Levinson. Like R's pacf().

```
my @pa = @{pacf([1,3,2,4,3,5], 3)}
```

pacf_basis

`pacf_basis` — batch 78 builtin.

```
my $result = pacf_basis $x
# or in a pipeline:
@list |> map pacf_basis |> p
```

packrat_parse_step

`packrat_parse_step` — batch 68 builtin.

```
my $result = packrat_parse_step $x
# or in a pipeline:
@list |> map packrat_parse_step |> p
```

pad_left

`pad_left` — trivial string ops builtin.

```
my $result = pad_left $x
# or in a pipeline:
@list |> map pad_left |> p
```

pad_number

`pad_number` — file stat / path builtin.

```
my $result = pad_number $x
# or in a pipeline:
@list |> map pad_number |> p
```

pad_right

`pad_right` — trivial string ops builtin.

```
my $result = pad_right $x
# or in a pipeline:
@list |> map pad_right |> p
```

page_size

`page_size` — memory page size in bytes. Uses `sysconf(_SC_PAGESIZE)` on unix.

```
p page_size                             # 16384 (Apple Silicon)
p page_size                             # 4096 (x86_64)
```

pagerank_damped

`pagerank_damped` — batch 24 builtin.

```
my $result = pagerank_damped $x
# or in a pipeline:
@list |> map pagerank_damped |> p
```

pagerank_iter

`pagerank_iter` — batch 75 builtin.

```
my $result = pagerank_iter $x
# or in a pipeline:
@list |> map pagerank_iter |> p
```

paired_ttest

`paired_ttest` (alias `pairedt`) performs a paired t-test on two matched samples. Returns [t-statistic, degrees of freedom].

```
my ($t, $df) = @{pairedt([85,90,78], [88,92,80])}
```

pairs_from_hash

`pairs_from_hash` — hash ops (batch 2) builtin.

```
my $result = pairs_from_hash $x
# or in a pipeline:
@list |> map pairs_from_hash |> p
```

pairwise

`pairwise` — file stat / path builtin.

```
my $result = pairwise $x
# or in a pipeline:
@list |> map pairwise |> p
```

pairwise_iter

`pairwise_iter` — python/ruby stdlib builtin.

```
my $result = pairwise_iter $x
# or in a pipeline:
@list |> map pairwise_iter |> p
```

palindromic_q

palindromic_q — batch 13 builtin.

```
my $result = palindromic_q $x
# or in a pipeline:
@list |> map palindromic_q |> p
```

pall

pall — algebraic match builtin. Alias for partition_all.

```
my $result = pall $x
# or in a pipeline:
@list |> map pall |> p
```

pam250_score

pam250_score — batch 33 builtin.

```
my $result = pam250_score $x
# or in a pipeline:
@list |> map pam250_score |> p
```

panel_between_estimator

panel_between_estimator — batch 36 builtin.

```
my $result = panel_between_estimator $x
# or in a pipeline:
@list |> map panel_between_estimator |> p
```

panel_random_effects

panel_random_effects — batch 36 builtin.

```
my $result = panel_random_effects $x
# or in a pipeline:
@list |> map panel_random_effects |> p
```

panel_var_step

panel_var_step — batch 36 builtin.

```
my $result = panel_var_step $x
# or in a pipeline:
@list |> map panel_var_step |> p
```

panel_within_estimator

panel_within_estimator — batch 36 builtin.

```
my $result = panel_within_estimator $x
# or in a pipeline:
@list |> map panel_within_estimator |> p
```

parabolic_d0

parabolic_d0 — batch 29 builtin.

```
my $result = parabolic_d0 $x
# or in a pipeline:
@list |> map parabolic_d0 |> p
```

parabolic_d1

`parabolic_d1` — batch 29 builtin.

```
my $result = parabolic_d1 $x
# or in a pipeline:
@list |> map parabolic_d1 |> p
```

parallax_correction

`parallax_correction` — batch 55 builtin.

```
my $result = parallax_correction $x
# or in a pipeline:
@list |> map parallax_correction |> p
```

parallel_machine_lpt

`parallel_machine_lpt` — batch 41 builtin.

```
my $result = parallel_machine_lpt $x
# or in a pipeline:
@list |> map parallel_machine_lpt |> p
```

parallel_machine_spt

`parallel_machine_spt` — batch 41 builtin.

```
my $result = parallel_machine_spt $x
# or in a pipeline:
@list |> map parallel_machine_spt |> p
```

parallel_transport_step

`parallel_transport_step` — batch 39 builtin.

```
my $result = parallel_transport_step $x
# or in a pipeline:
@list |> map parallel_transport_step |> p
```

parenthesize

`parenthesize` — string helpers (batch 4) builtin.

```
my $result = parenthesize $x
# or in a pipeline:
@list |> map parenthesize |> p
```

pareto_dominance

`pareto_dominance` — batch 73 builtin.

```
my $result = pareto_dominance $input
```

pareto_efficiency

`pareto_efficiency` — batch 73 builtin.

```
my $result = pareto_efficiency $input
```

pareto_pdf

`pareto_pdf` (alias `paretopdf`) evaluates the Pareto distribution PDF at x with minimum xm and shape alpha.

```
p paretopdf(2, 1, 3)  # Pareto(1,3) at x=2
```

parse_bool

`parse_bool` — extended stdlib builtin.

```
my $result = parse_bool $x
# or in a pipeline:
@list |> map parse_bool |> p
```

parse_float

`parse_float` — extended stdlib builtin.

```
my $result = parse_float $x
# or in a pipeline:
@list |> map parse_float |> p
```

parse_int

`parse_int` — extended stdlib builtin.

```
my $result = parse_int $x
# or in a pipeline:
@list |> map parse_int |> p
```

partial_pressure

`partial_pressure` — batch 12 builtin.

```
my $result = partial_pressure $x
# or in a pipeline:
@list |> map partial_pressure |> p
```

particle_filter_resample

`particle_filter_resample` — batch 40 builtin.

```
my $result = particle_filter_resample $x
# or in a pipeline:
@list |> map particle_filter_resample |> p
```

partition_all

`partition_all` — algebraic match builtin.

```
my $result = partition_all $x
# or in a pipeline:
@list |> map partition_all |> p
```

partition_at

`partition_at` — batch 9 builtin.

```
my $result = partition_at $x
# or in a pipeline:
@list |> map partition_at |> p
```

partition_by

`partition_by` — algebraic match builtin.

```
my $result = partition_by $x
# or in a pipeline:
@list |> map partition_by |> p
```

partition_either

`partition_either` — rust iterator methods builtin.

```
my $result = partition_either $x
# or in a pipeline:
@list |> map partition_either |> p
```

partition_n

`partition_n` — go/general functional utilities builtin.

```
my $result = partition_n $x
# or in a pipeline:
@list |> map partition_n |> p
```

partition_number

`partition_number` — math / numeric (uncategorized batch) builtin.

```
my $result = partition_number $x
# or in a pipeline:
@list |> map partition_number |> p
```

partition_point

`partition_point` — extended stdlib builtin.

```
my $result = partition_point $x
# or in a pipeline:
@list |> map partition_point |> p
```

partition_two

`partition_two` — list helpers (batch 4) builtin.

```
my $result = partition_two $x
# or in a pipeline:
@list |> map partition_two |> p
```

partn

`partn` — go/general functional utilities builtin. Alias for `partition_n`.

```
my $result = partn $x
# or in a pipeline:
@list |> map partn |> p
```

partn_num

`partn_num` — math / numeric (uncategorized batch) builtin. Alias for `partition_number`.

```
my $result = partn_num $x
# or in a pipeline:
@list |> map partn_num |> p
```

parzen_window

`parzen_window` — batch 74 builtin.

```
my $result = parzen_window $x
# or in a pipeline:
@list |> map parzen_window |> p
```

pascal_case

`pascal_case` — string (batch 2) builtin.

```
my $result = pascal_case $x
# or in a pipeline:
@list |> map pascal_case |> p
```

pascal_row

`pascal_row` — extended stdlib batch 3 builtin.

```
my $result = pascal_row $x
# or in a pipeline:
@list |> map pascal_row |> p
```

pascals_to_bar

`pascals_to_bar` — file stat / path builtin.

```
my $result = pascals_to_bar $input
```

pascals_to_psi

`pascals_to_psi` — file stat / path builtin.

```
my $result = pascals_to_psi $input
```

pascals_triangle

`pascals_triangle` — math / numeric (uncategorized batch) builtin.

```
my $result = pascals_triangle $x
# or in a pipeline:
@list |> map pascals_triangle |> p
```

pascrow

`pascrow` — extended stdlib batch 3 builtin. Alias for `pascal_row`.

```
my $result = pascrow $x
# or in a pipeline:
@list |> map pascrow |> p
```

pasctri

`pasctri` — math / numeric (uncategorized batch) builtin. Alias for `pascals_triangle`.

```
my $result = pasctri $x
# or in a pipeline:
@list |> map pasctri |> p
```

pass

`pass` — conversion / utility (batch 4) builtin.

```
my $result = pass $x
# or in a pipeline:
@list |> map pass |> p
```

path_case

`path_case` — string (batch 2) builtin.

```
my $result = path_case $x
# or in a pipeline:
@list |> map path_case |> p
```

path_dubins_lsl

`path_dubins_lsl` — batch 59 builtin.

```
my $result = path_dubins_lsl $x
# or in a pipeline:
@list |> map path_dubins_lsl |> p
```

path_dubins_rsr

`path_dubins_rsr` — batch 59 builtin.

```
my $result = path_dubins_rsr $x
# or in a pipeline:
@list |> map path_dubins_rsr |> p
```

path_ext

`path_ext` — path helpers builtin.

```
my $result = path_ext $x
# or in a pipeline:
@list |> map path_ext |> p
```

path_is_abs

`path_is_abs` — file stat / path builtin.

```
my $result = path_is_abs $x
# or in a pipeline:
@list |> map path_is_abs |> p
```

path_is_rel

`path_is_rel` — file stat / path builtin.

```
my $result = path_is_rel $x
# or in a pipeline:
@list |> map path_is_rel |> p
```

path_join

`path_join` — path helpers builtin.

```
my $result = path_join $x
# or in a pipeline:
@list |> map path_join |> p
```

path_parent

`path_parent` — path helpers builtin.

```
my $result = path_parent $x
# or in a pipeline:
@list |> map path_parent |> p
```

path_reeds_shepp

`path_reeds_shepp` — batch 59 builtin.

```
my $result = path_reeds_shepp $x
# or in a pipeline:
@list |> map path_reeds_shepp |> p
```

path_split

`path_split` — path helpers builtin.

```
my $result = path_split $x
# or in a pipeline:
@list |> map path_split |> p
```

path_stem

`path_stem` — path helpers builtin.

```
my $result = path_stem $x
# or in a pipeline:
@list |> map path_stem |> p
```

pauli_decomposition_2x2

`pauli_decomposition_2x2` — batches 14-16 builtin.

```
my $result = pauli_decomposition_2x2 $x
# or in a pipeline:
@list |> map pauli_decomposition_2x2 |> p
```

pauli_real_part

`pauli_real_part` — batch 32 builtin.

```
my $result = pauli_real_part $x
# or in a pipeline:
@list |> map pauli_real_part |> p
```

pauli_string_expect

pauli_string_expect — batch 80 builtin.

```
my $result = pauli_string_expect $x
# or in a pipeline:
@list |> map pauli_string_expect |> p
```

pauli_y_imag

pauli_y_imag — batch 32 builtin.

```
my $result = pauli_y_imag $x
# or in a pipeline:
@list |> map pauli_y_imag |> p
```

payback

payback_period($initial, @cashflows) (alias payback) — computes the number of periods to recover initial investment. Returns fractional periods.

```
p payback(1000, 300, 400, 500)   # 2.6 periods
```

payback_period_simple

payback_period_simple — batch 10 builtin.

```
my $result = payback_period_simple $x
# or in a pipeline:
@list |> map payback_period_simple |> p
```

payback_simple

payback_simple — batch 10 builtin. Alias for payback_period_simple.

```
my $result = payback_simple $x
# or in a pipeline:
@list |> map payback_simple |> p
```

pbeta

pbeta — beta CDF (regularized incomplete beta). Args: x, a, b.

```
p pbeta(0.5, 2, 5)  # P(X ≤ 0.5) for Beta(2,5)
```

pbinom

pbinom — binomial CDF. P(X ≤ k) for Binom(n, p). Args: k, n, p.

```
p pbinom(5, 10, 0.5)   # P(≤5 heads in 10 flips)
```

pbkdf2_hmac_step

pbkdf2_hmac_step — batch 26 builtin.

```
my $result = pbkdf2_hmac_step $x
# or in a pipeline:
@list |> map pbkdf2_hmac_step |> p
```

pbool

`pbool` — extended stdlib builtin. Alias for `parse_bool`.

```
my $result = pbool $x
# or in a pipeline:
@list |> map pbool |> p
```

pby

`pby` — algebraic match builtin. Alias for `partition_by`.

```
my $result = pby $x
# or in a pipeline:
@list |> map pby |> p
```

pc_case

`pc_case` — string (batch 2) builtin. Alias for `pascal_case`.

```
my $result = pc_case $x
# or in a pipeline:
@list |> map pc_case |> p
```

pc_to_au

`pc_to_au` — batch 30 builtin.

```
my $result = pc_to_au $input
```

pc_to_ly

`pc_to_ly` — batch 30 builtin.

```
my $result = pc_to_ly $input
```

pca

`prcomp` (alias `pca`) — Principal Component Analysis via eigendecomposition of covariance matrix. Returns eigenvalues (variance explained). Like R's prcomp().

```
my @var = @{pca([[1,2],[3,4],[5,6],[7,8]])}
# variance explained by each component
```

pcauchy

`pcauchy` — Cauchy CDF. P(X ≤ x). Args: x [, location, scale].

```
p pcauchy(0, 0, 1)  # 0.5
```

pcg32_step

`pcg32_step` — batch 26 builtin.

```
my $result = pcg32_step $x
# or in a pipeline:
@list |> map pcg32_step |> p
```

pchisq

`pchisq` — chi-squared CDF. Args: x, df.

```
p pchisq(3.84, 1)  # 0.95 (critical value for p=0.05)
```

pcr

`pcr` — data / network builtin. Alias for `par_csv_read`.

```
my $result = pcr $x
# or in a pipeline:
@list |> map pcr |> p
```

pct

`pct` — trivial numeric / predicate builtins builtin. Alias for `percent`.

```
my $result = pct $x
# or in a pipeline:
@list |> map pct |> p
```

pctrank

`pctrank` — statistics (extended) builtin. Alias for `percentile_rank`.

```
my $result = pctrank $x
# or in a pipeline:
@list |> map pctrank |> p
```

pdo_index_step

`pdo_index_step` — batch 42 builtin.

```
my $result = pdo_index_step $x
# or in a pipeline:
@list |> map pdo_index_step |> p
```

peak_detect

`peak_detect` — matrix operations (uncategorized batch) builtin.

```
my $result = peak_detect $x
# or in a pipeline:
@list |> map peak_detect |> p
```

peak_widths_at

`peak_widths_at` — batch 74 builtin.

```
my $result = peak_widths_at $x
# or in a pipeline:
@list |> map peak_widths_at |> p
```

peaks

`peaks` — matrix operations (uncategorized batch) builtin. Alias for `peak_detect`.

```
my $result = peaks $x
# or in a pipeline:
@list |> map peaks |> p
```

pearson_hash

`pearson_hash` — batch 9 builtin. Alias for `pearson_hash_byte`.

```
my $result = pearson_hash $x
# or in a pipeline:
@list |> map pearson_hash |> p
```

pearson_hash_byte

`pearson_hash_byte` — batch 9 builtin.

```
my $result = pearson_hash_byte $x
# or in a pipeline:
@list |> map pearson_hash_byte |> p
```

pearson_skewness_2

`pearson_skewness_2` — batches 14-16 builtin.

```
my $result = pearson_skewness_2 $x
# or in a pipeline:
@list |> map pearson_skewness_2 |> p
```

pearsonr

`pearsonr` — batch 53 builtin.

```
my $result = pearsonr $x
# or in a pipeline:
@list |> map pearsonr |> p
```

peclet_number_step

`peclet_number_step` — batch 42 builtin.

```
my $result = peclet_number_step $x
# or in a pipeline:
@list |> map peclet_number_step |> p
```

peek_clj

`peek_clj` — algebraic match builtin.

```
my $result = peek_clj $x
# or in a pipeline:
@list |> map peek_clj |> p
```

peeling_step_psi4

`peeling_step_psi4` — batch 39 builtin.

```
my $result = peeling_step_psi4 $x
# or in a pipeline:
@list |> map peeling_step_psi4 |> p
```

peg_choice

`peg_choice` — batch 68 builtin.

```
my $result = peg_choice $x
# or in a pipeline:
@list |> map peg_choice |> p
```

peg_lookahead

`peg_lookahead` — batch 68 builtin.

```
my $result = peg_lookahead $x
# or in a pipeline:
@list |> map peg_lookahead |> p
```

peg_repeat

`peg_repeat` — batch 68 builtin.

```
my $result = peg_repeat $x
# or in a pipeline:
@list |> map peg_repeat |> p
```

peg_seq

`peg_seq` — batch 68 builtin.

```
my $result = peg_seq $x
# or in a pipeline:
@list |> map peg_seq |> p
```

peith

`peith` — rust iterator methods builtin. Alias for `partition_either`.

```
my $result = peith $x
# or in a pipeline:
@list |> map peith |> p
```

pell_fundamental

`pell_fundamental` — batch 35 builtin.

```
my $result = pell_fundamental $x
# or in a pipeline:
@list |> map pell_fundamental |> p
```

pendper

`pendulum_period($length)` (alias `pendper`) — period T = 2π√(L/g) of a simple pendulum.

```
p pendper(1)      # ~2.01 s (1 meter pendulum)
p pendper(0.25)   # ~1.00 s (grandfather clock)
```

pendulum_freq

`pendulum_freq` — batch 12 builtin.

```
my $result = pendulum_freq $x
# or in a pipeline:
@list |> map pendulum_freq |> p
```

penrose_diagram_factor

`penrose_diagram_factor` — batch 39 builtin.

```
my $result = penrose_diagram_factor $x
# or in a pipeline:
@list |> map penrose_diagram_factor |> p
```

pentagonal_number

`pentagonal_number` — number theory / primes builtin.

```
my $result = pentagonal_number $x
# or in a pipeline:
@list |> map pentagonal_number |> p
```

pentagonal_pyramidal_number

`pentagonal_pyramidal_number` — batch 10 builtin.

```
my $result = pentagonal_pyramidal_number $x
# or in a pipeline:
@list |> map pentagonal_pyramidal_number |> p
```

pentnum

`pentnum` — number theory / primes builtin. Alias for `pentagonal_number`.

```
my $result = pentnum $x
# or in a pipeline:
@list |> map pentnum |> p
```

percent

`percent` — trivial numeric / predicate builtins builtin.

```
my $result = percent $x
# or in a pipeline:
@list |> map percent |> p
```

percent_decode_url

`percent_decode_url` — batch 81 builtin.

```
my $result = percent_decode_url $x
# or in a pipeline:
@list |> map percent_decode_url |> p
```

percent_encode_full

`percent_encode_full` — batch 62 builtin.

```
my $result = percent_encode_full $x
# or in a pipeline:
@list |> map percent_encode_full |> p
```

percentile

`percentile` — file stat / path builtin.

```
my $result = percentile $x
# or in a pipeline:
@list |> map percentile |> p
```

percentile_cont

`percentile_cont` — batch 70 builtin.

```
my $result = percentile_cont $x
# or in a pipeline:
@list |> map percentile_cont |> p
```

percentile_disc

`percentile_disc` — batch 70 builtin.

```
my $result = percentile_disc $x
# or in a pipeline:
@list |> map percentile_disc |> p
```

percentile_rank

`percentile_rank($value, @data)` — returns the percentile rank (0-100) of a value within a dataset. Indicates what percentage of the data falls below the given value. Useful for ranking and normalization.

```
my @scores = 1:100
p percentile_rank(50, @scores)   # ~50
p percentile_rank(90, @scores)   # ~90
```

percentrank

`percentrank` — batch 57 builtin.

```
my $result = percentrank $x
# or in a pipeline:
@list |> map percentrank |> p
```

perfect_numbers

`perfect_numbers` — number theory / primes builtin.

```
my $result = perfect_numbers $x
# or in a pipeline:
@list |> map perfect_numbers |> p
```

perfnums

`perfnums` — number theory / primes builtin. Alias for `perfect_numbers`.

```
my $result = perfnums $x
# or in a pipeline:
@list |> map perfnums |> p
```

perihelion_precession

`perihelion_precession` — batch 31 builtin.

```
my $result = perihelion_precession $x
# or in a pipeline:
@list |> map perihelion_precession |> p
```

perimeter_rectangle

`perimeter_rectangle` — geometry / physics builtin.

```
my $result = perimeter_rectangle $x
# or in a pipeline:
@list |> map perimeter_rectangle |> p
```

perimeter_triangle

`perimeter_triangle` — geometry / physics builtin.

```
my $result = perimeter_triangle $x
# or in a pipeline:
@list |> map perimeter_triangle |> p
```

period_of_string

`period_of_string` — batches 14-16 builtin.

```
my $result = period_of_string $x
# or in a pipeline:
@list |> map period_of_string |> p
```

periodogram_basic

`periodogram_basic` — batch 74 builtin.

```
my $result = periodogram_basic $x
# or in a pipeline:
@list |> map periodogram_basic |> p
```

perm

`permutations($n, $r)` (alias `perm`) — number of ways to arrange r items from n. Formula: n!/(n-r)!. Order matters.

```
p perm(5, 3)    # 60 (ways to arrange 3 items from 5)
p perm(10, 2)   # 90
```

perm_mul

`perm_mul` — batch 9 builtin. Alias for `permutation_compose`.

```
my $result = perm_mul $x
# or in a pipeline:
@list |> map perm_mul |> p
```

perm_sign

`perm_sign` — batch 9 builtin. Alias for `permutation_parity`.

```
my $result = perm_sign $x
# or in a pipeline:
@list |> map perm_sign |> p
```

permutation_compose

`permutation_compose` — batch 9 builtin.

```
my $result = permutation_compose $x
# or in a pipeline:
@list |> map permutation_compose |> p
```

permutation_order

`permutation_order` — batch 9 builtin.

```
my $result = permutation_order $x
# or in a pipeline:
@list |> map permutation_order |> p
```

permutation_parity

`permutation_parity` — batch 9 builtin.

```
my $result = permutation_parity $x
# or in a pipeline:
@list |> map permutation_parity |> p
```

permute_idx

`permute_idx` — batch 54 builtin.

```
my $result = permute_idx $x
# or in a pipeline:
@list |> map permute_idx |> p
```

perpetuity_value

`perpetuity_value` — batch 13 builtin.

```
my $result = perpetuity_value $x
# or in a pipeline:
@list |> map perpetuity_value |> p
```

persian_arithmetic_leap

`persian_arithmetic_leap` — batch 52 builtin.

```
my $result = persian_arithmetic_leap $x
# or in a pipeline:
@list |> map persian_arithmetic_leap |> p
```

persist

`persist` — batch 71 builtin.

```
my $result = persist $x
# or in a pipeline:
@list |> map persist |> p
```

personalized_pagerank

`personalized_pagerank` — batch 75 builtin.

```
my $result = personalized_pagerank $x
# or in a pipeline:
@list |> map personalized_pagerank |> p
```

petersen_estimator

petersen_estimator — batch 22 builtin.

```
my $result = petersen_estimator $x
# or in a pipeline:
@list |> map petersen_estimator |> p
```

peukert_capacity

peukert_capacity — batch 38 builtin.

```
my $result = peukert_capacity $x
# or in a pipeline:
@list |> map peukert_capacity |> p
```

peukert_exponent_fit

peukert_exponent_fit — batch 38 builtin.

```
my $result = peukert_exponent_fit $x
# or in a pipeline:
@list |> map peukert_exponent_fit |> p
```

pexp

pexp — exponential CDF. P(X ? x) for Exp(rate). Args: x, rate.

```
p pexp(1, 1)     # 0.6321 (1 - e^-1)
p pexp(5, 0.5)   # P(wait ≤ 5 with avg wait 2)
```

pf

pf — F-distribution CDF. Args: x, d1, d2.

```
p pf(4.0, 5, 10)  # P(F ≤ 4) for F(5,10)
```

pfact

pfact — number theory / primes builtin. Alias for prime_factors.

```
my $result = pfact $x
# or in a pipeline:
@list |> map pfact |> p
```

pfadd

pfadd — batch 71 builtin.

```
my $result = pfadd $x
# or in a pipeline:
@list |> map pfadd |> p
```

pfcount

pfcount — batch 71 builtin.

```
my $result = pfcount $x
# or in a pipeline:
@list |> map pfcount |> p
```

pflt

`pflt` — extended stdlib builtin. Alias for `parse_float`.

```
my $result = pflt $x
# or in a pipeline:
@list |> map pflt |> p
```

pfreq

`pfreq` — pipeline / string helpers builtin. Alias for `pfrequencies`.

```
my $result = pfreq $x
# or in a pipeline:
@list |> map pfreq |> p
```

pfrequencies

`pfrequencies` — pipeline / string helpers builtin.

```
my $result = pfrequencies $x
# or in a pipeline:
@list |> map pfrequencies |> p
```

pfrq

`pfrq` — pipeline / string helpers builtin. Alias for `pfrequencies`.

```
my $result = pfrq $x
# or in a pipeline:
@list |> map pfrq |> p
```

pft

`pft` — data / network builtin. Alias for `par_fetch`.

```
my $result = pft $x
# or in a pipeline:
@list |> map pft |> p
```

pga_attenuation

`pga_attenuation` — batch 65 builtin.

```
my $result = pga_attenuation $x
# or in a pipeline:
@list |> map pga_attenuation |> p
```

pgamma

`pgamma` — gamma CDF. Args: x, shape [, scale].

```
p pgamma(2, 2, 1)  # P(X ≤ 2) for Gamma(2,1)
```

ph_at_isoelectric

`ph_at_isoelectric` — batch 38 builtin.

```
my $result = ph_at_isoelectric $x
# or in a pipeline:
@list |> map ph_at_isoelectric |> p
```

ph_from_h

`ph_from_h` — batch 21 builtin.

```
my $result = ph_from_h $x
# or in a pipeline:
@list |> map ph_from_h |> p
```

phase_flip_prob

`phase_flip_prob` — batch 32 builtin.

```
my $result = phase_flip_prob $x
# or in a pipeline:
@list |> map phase_flip_prob |> p
```

phase_spectrum

`phase_spectrum(\@signal)` — computes the phase angle (in radians) at each frequency bin from the DFT.

```
my $phases = phase_spectrum(\@signal)
```

phasespec

`phasespec` — dsp / signal (extended) builtin. Alias for `phase_spectrum`.

```
my $result = phasespec $x
# or in a pipeline:
@list |> map phasespec |> p
```

phillips_perron

`phillips_perron` — batch 78 builtin.

```
my $result = phillips_perron $x
# or in a pipeline:
@list |> map phillips_perron |> p
```

phonedigit

`phonedigit` — encoding / phonetics builtin. Alias for `phonetic_digit`.

```
my $result = phonedigit $x
# or in a pipeline:
@list |> map phonedigit |> p
```

phonetic_digit

`phonetic_digit` — encoding / phonetics builtin.

```
my $result = phonetic_digit $x
# or in a pipeline:
@list |> map phonetic_digit |> p
```

photocurrent_density

`photocurrent_density` — batch 38 builtin.

```
my $result = photocurrent_density $x
# or in a pipeline:
@list |> map photocurrent_density |> p
```

photon_energy

`photon_energy($frequency)` (alias `photonenergy`) — energy E = hf (Joules) of a photon.

```
p photonenergy(5e14)   # ~3.3e-19 J (visible light)
```

photon_energy_freq

`photon_energy_freq` — batch 21 builtin.

```
my $result = photon_energy_freq $x
# or in a pipeline:
@list |> map photon_energy_freq |> p
```

photon_energy_lambda

`photon_energy_lambda` — batch 21 builtin.

```
my $result = photon_energy_lambda $x
# or in a pipeline:
@list |> map photon_energy_lambda |> p
```

photon_energy_wavelength

`photon_energy_wavelength` — physics formulas builtin.

```
my $result = photon_energy_wavelength $x
# or in a pipeline:
@list |> map photon_energy_wavelength |> p
```

photon_sphere_radius

`photon_sphere_radius` — batch 31 builtin.

```
my $result = photon_sphere_radius $x
# or in a pipeline:
@list |> map photon_sphere_radius |> p
```

photonlambda

`photonlambda` — physics formulas builtin. Alias for `photon_energy_wavelength`.

```
my $result = photonlambda $x
# or in a pipeline:
@list |> map photonlambda |> p
```

pi

`pi` — constants builtin.

```
my $result = pi $x
# or in a pipeline:
@list |> map pi |> p
```

pi_step_size

`pi_step_size` — batch 34 builtin.

```
my $result = pi_step_size $x
# or in a pipeline:
@list |> map pi_step_size |> p
```

pick

`pick` — hash ops (batch 2) builtin. Alias for `pick_keys`.

```
my $result = pick $x
# or in a pipeline:
@list |> map pick |> p
```

pick_keys

`pick_keys` — hash ops (batch 2) builtin.

```
my $result = pick_keys $x
# or in a pipeline:
@list |> map pick_keys |> p
```

picks_theorem

`picks_theorem` — batch 28 builtin.

```
my $result = picks_theorem $x
# or in a pipeline:
@list |> map picks_theorem |> p
```

pid

`pid` — system introspection builtin.

```
my $result = pid $x
# or in a pipeline:
@list |> map pid |> p
```

pid_anti_windup

`pid_anti_windup` — batch 59 builtin.

```
my $result = pid_anti_windup $x
# or in a pipeline:
@list |> map pid_anti_windup |> p
```

pid_ziegler_nichols

`pid_ziegler_nichols` — batch 59 builtin.

```
my $result = pid_ziegler_nichols $x
# or in a pipeline:
@list |> map pid_ziegler_nichols |> p
```

pielou_evenness

`pielou_evenness` — batch 22 builtin.

```
my $result = pielou_evenness $x
# or in a pipeline:
@list |> map pielou_evenness |> p
```

pig_latin

`pig_latin` — string processing (uncategorized batch) builtin.

```
my $result = pig_latin $x
# or in a pipeline:
@list |> map pig_latin |> p
```

piglat

`piglat` — string processing (uncategorized batch) builtin. Alias for `pig_latin`.

```
my $result = piglat $x
# or in a pipeline:
@list |> map piglat |> p
```

pin

`pin` — stress testing builtin. Alias for `fire_and_forget`.

```
my $result = pin $x
# or in a pipeline:
@list |> map pin |> p
```

pint

`pint` — extended stdlib builtin. Alias for `parse_int`.

```
my $result = pint $x
# or in a pipeline:
@list |> map pint |> p
```

pipe_pressure_drop

`pipe_pressure_drop` — batch 13 builtin.

```
my $result = pipe_pressure_drop $x
# or in a pipeline:
@list |> map pipe_pressure_drop |> p
```

pipes

`pipes` — filesystem extensions builtin.

```
my $result = pipes $x
# or in a pipeline:
@list |> map pipes |> p
```

pivoted_cholesky_var

`pivoted_cholesky_var` — batch 78 builtin.

```
my $result = pivoted_cholesky_var $x
# or in a pipeline:
@list |> map pivoted_cholesky_var |> p
```

pka_from_ka

`pka_from_ka` — batch 21 builtin.

```
my $result = pka_from_ka $x
# or in a pipeline:
@list |> map pka_from_ka |> p
```

pkb_to_kb

`pkb_to_kb` — batch 63 builtin.

```
my $result = pkb_to_kb $input
```

pkc

`pkc` — algebraic match builtin. Alias for `peek_clj`.

```
my $result = pkc $x
# or in a pipeline:
@list |> map pkc |> p
```

pkcs7_pad

`pkcs7_pad` — batches 17-18 builtin.

```
my $result = pkcs7_pad $x
# or in a pipeline:
@list |> map pkcs7_pad |> p
```

pkcs7_unpad

`pkcs7_unpad` — batches 17-18 builtin.

```
my $result = pkcs7_unpad $x
# or in a pipeline:
@list |> map pkcs7_unpad |> p
```

plactic_class_size

`plactic_class_size` — batch 37 builtin.

```
my $result = plactic_class_size $x
# or in a pipeline:
@list |> map plactic_class_size |> p
```

planck

`planck` — math / physics constants builtin.

```
my $result = planck $x
# or in a pipeline:
@list |> map planck |> p
```

planck_const_h

`planck_const_h` — batch 12 builtin.

```
my $result = planck_const_h $x
# or in a pipeline:
@list |> map planck_const_h |> p
```

planck_constant

`planck_constant` — constants builtin.

```
my $result = planck_constant $x
# or in a pipeline:
@list |> map planck_constant |> p
```

planck_energy

`planck_energy` — batch 31 builtin.

```
my $result = planck_energy $x
# or in a pipeline:
@list |> map planck_energy |> p
```

planck_mass

`planck_mass` — physics constants builtin.

```
my $result = planck_mass $x
# or in a pipeline:
@list |> map planck_mass |> p
```

planck_spectral_radiance

`planck_spectral_radiance` — batch 31 builtin.

```
my $result = planck_spectral_radiance $x
# or in a pipeline:
@list |> map planck_spectral_radiance |> p
```

planck_temperature

`planck_temperature` — physics constants builtin.

```
my $result = planck_temperature $x
# or in a pipeline:
@list |> map planck_temperature |> p
```

planck_time

`planck_time` — physics constants builtin.

```
my $result = planck_time $x
# or in a pipeline:
@list |> map planck_time |> p
```

plastic

`plastic` — math constants builtin. Alias for `plastic_number`.

```
my $result = plastic $x
# or in a pipeline:
@list |> map plastic |> p
```

plastic_number

`plastic_number` — math constants builtin.

```
my $result = plastic_number $x
# or in a pipeline:
@list |> map plastic_number |> p
```

plethystic_substitution_value

`plethystic_substitution_value` — batch 37 builtin.

```
my $result = plethystic_substitution_value $x
# or in a pipeline:
@list |> map plethystic_substitution_value |> p
```

plnorm

`plnorm` — log-normal CDF. P(X ? x) for LogN(mu, sigma). Args: x, mu, sigma.

```
p plnorm(1, 0, 1)   # 0.5 (median of LogN(0,1))
```

plur

`plur` — extended stdlib builtin. Alias for `pluralize`.

```
my $result = plur $x
# or in a pipeline:
@list |> map plur |> p
```

plurality_winner_step

`plurality_winner_step` — batch 43 builtin.

```
my $result = plurality_winner_step $x
# or in a pipeline:
@list |> map plurality_winner_step |> p
```

pluralize

`pluralize` — extended stdlib builtin.

```
my $result = pluralize $x
# or in a pipeline:
@list |> map pluralize |> p
```

pluralize_simple

`pluralize_simple` — batch 9 builtin.

```
my $result = pluralize_simple $x
# or in a pipeline:
@list |> map pluralize_simple |> p
```

plus_polarization_amp

`plus_polarization_amp` — batch 39 builtin.

```
my $result = plus_polarization_amp $x
# or in a pipeline:
@list |> map plus_polarization_amp |> p
```

pmass

`pmass` — constants builtin. Alias for `proton_mass`.

```
my $result = pmass $x
# or in a pipeline:
@list |> map pmass |> p
```

pmt

`pmt($rate, $nper, $pv)` — computes the periodic payment for a loan given interest rate, number of periods, and present value. Standard amortization formula.

```
# $100k loan, 5% annual rate, 30 years
p pmt(0.05/12, 360, 100000)   # ~$537/month
```

pnorm

`pnorm` — normal CDF. P(X ≤ x) for N(mu, sigma). Args: x [, mu, sigma]. Default standard normal.

```
p pnorm(0)          # 0.5
p pnorm(1.96)       # 0.975
p pnorm(100, 90, 15) # z-score for IQ 100
```

pochhammer

`pochhammer` — batch 29 builtin.

```
my $result = pochhammer $x
# or in a pipeline:
@list |> map pochhammer |> p
```

poh_from_oh

`poh_from_oh` — batch 21 builtin.

```
my $result = poh_from_oh $x
# or in a pipeline:
@list |> map poh_from_oh |> p
```

poincare_duality

`poincare_duality` — batch 37 builtin.

```
my $result = poincare_duality $x
# or in a pipeline:
@list |> map poincare_duality |> p
```

poincare_polynomial_eval

`poincare_polynomial_eval` — batch 37 builtin.

```
my $result = poincare_polynomial_eval $x
# or in a pipeline:
@list |> map poincare_polynomial_eval |> p
```

point_distance

`point_distance` — geometry / physics builtin.

```
my $result = point_distance $x
# or in a pipeline:
@list |> map point_distance |> p
```

point_in_polygon

`point_in_polygon($x, $y, @polygon)` (alias `pip`) — tests if a point lies inside a polygon. Polygon is given as flat list [x1,y1,x2,y2,...]. Returns 1 if inside, 0 if outside. Uses ray-casting algorithm.

```
my @square = (0,0, 4,0, 4,4, 0,4)
p pip(2, 2, @square)   # 1 (inside)
p pip(5, 5, @square)   # 0 (outside)
```

point_in_polygon_ray

point_in_polygon_ray — batch 58 builtin.

```
my $result = point_in_polygon_ray $x
# or in a pipeline:
@list |> map point_in_polygon_ray |> p
```

point_in_polygon_winding

point_in_polygon_winding — batch 58 builtin.

```
my $result = point_in_polygon_winding $x
# or in a pipeline:
@list |> map point_in_polygon_winding |> p
```

pointer_width

pointer_width — pointer width in bits: 32 or 64. Compile-time constant.

```
p pointer_width                        # 64
```

poisson

poisson_pmf($k, $lambda) (alias poisson) — Poisson probability mass function. P(X=k) for events with rate λ.

```
p poisson(3, 5)   # P(X=3) when avg rate is 5
```

poisson_boltzmann_step

poisson_boltzmann_step — batch 38 builtin.

```
my $result = poisson_boltzmann_step $x
# or in a pipeline:
@list |> map poisson_boltzmann_step |> p
```

poisson_photon_pmf

poisson_photon_pmf — batch 32 builtin.

```
my $result = poisson_photon_pmf $x
# or in a pipeline:
@list |> map poisson_photon_pmf |> p
```

polar_vortex_radius

polar_vortex_radius — batch 42 builtin.

```
my $result = polar_vortex_radius $x
# or in a pipeline:
@list |> map polar_vortex_radius |> p
```

policy_iteration_step

policy_iteration_step — batch 40 builtin.

```
my $result = policy_iteration_step $x
# or in a pipeline:
@list |> map policy_iteration_step |> p
```

pollard_p_minus_1

`pollard_p_minus_1` — batch 35 builtin.

```
my $result = pollard_p_minus_1 $x
# or in a pipeline:
@list |> map pollard_p_minus_1 |> p
```

poly1305_block_step

`poly1305_block_step` — batches 17-18 builtin.

```
my $result = poly1305_block_step $x
# or in a pipeline:
@list |> map poly1305_block_step |> p
```

poly_eval

`poly_eval` (alias `polyval`) evaluates a polynomial c0 + c1·x + c2·x² + ... using Horner's method.

```
p polyval([1, 0, 1], 3)  # 1 + 0*3 + 1*9 = 10
```

polyarea

`polyarea` — geometry / physics builtin. Alias for `polygon_area`.

```
my $result = polyarea $x
# or in a pipeline:
@list |> map polyarea |> p
```

polyfit

`polynomial_fit` (alias `polyfit`) performs least-squares polynomial fitting. Returns coefficients [c0, c1, ..., cn].

```
my $c = polyfit([0,1,2,3], [1,3,5,7], 1)  # linear fit
```

polyfit_rmse

`polyfit_rmse` — batch 72 builtin.

```
my $result = polyfit_rmse $x
# or in a pipeline:
@list |> map polyfit_rmse |> p
```

polygamma

`polygamma` — batch 72 builtin.

```
my $result = polygamma $x
# or in a pipeline:
@list |> map polygamma |> p
```

polygon_area

`polygon_area` — geometry / physics builtin.

```
my $result = polygon_area $x
# or in a pipeline:
@list |> map polygon_area |> p
```

polygon_area_signed

`polygon_area_signed` — batches 17-18 builtin.

```
my $result = polygon_area_signed $x
# or in a pipeline:
@list |> map polygon_area_signed |> p
```

polygon_convex_q

`polygon_convex_q` — batches 17-18 builtin.

```
my $result = polygon_convex_q $x
# or in a pipeline:
@list |> map polygon_convex_q |> p
```

polygon_is_convex

`polygon_is_convex` — batch 28 builtin.

```
my $result = polygon_is_convex $x
# or in a pipeline:
@list |> map polygon_is_convex |> p
```

polygon_perimeter

`polygon_perimeter(@points)` (alias `polyper`) — computes the perimeter of a polygon. Points as flat list [x1,y1,x2,y2,...]. Assumes closed polygon.

```
my @square = (0,0, 1,0, 1,1, 0,1)
p polyper(@square)   # 4
```

polygon_winding_number

`polygon_winding_number` — batches 17-18 builtin.

```
my $result = polygon_winding_number $x
# or in a pipeline:
@list |> map polygon_winding_number |> p
```

polygon_winding_order

`polygon_winding_order` — batch 58 builtin.

```
my $result = polygon_winding_order $x
# or in a pipeline:
@list |> map polygon_winding_order |> p
```

polygon_with_holes_area

`polygon_with_holes_area` — batch 28 builtin.

```
my $result = polygon_with_holes_area $x
# or in a pipeline:
@list |> map polygon_with_holes_area |> p
```

polylog_li2

`polylog_li2` — batch 29 builtin.

```
my $result = polylog_li2 $x
# or in a pipeline:
@list |> map polylog_li2 |> p
```

polylog_n

`polylog_n` — batch 29 builtin.

```
my $result = polylog_n $x
# or in a pipeline:
@list |> map polylog_n |> p
```

polynomial_jones_skein

`polynomial_jones_skein` — batch 37 builtin.

```
my $result = polynomial_jones_skein $x
# or in a pipeline:
@list |> map polynomial_jones_skein |> p
```

polynomial_roots_dk

`polynomial_roots_dk` — batch 10 builtin.

```
my $result = polynomial_roots_dk $x
# or in a pipeline:
@list |> map polynomial_roots_dk |> p
```

polyperim

`polyperim` — geometry (extended) builtin. Alias for `polygon_perimeter`.

```
my $result = polyperim $x
# or in a pipeline:
@list |> map polyperim |> p
```

pooling_test_stat

`pooling_test_stat` — batch 36 builtin.

```
my $result = pooling_test_stat $x
# or in a pipeline:
@list |> map pooling_test_stat |> p
```

pop_clj

`pop_clj` — algebraic match builtin.

```
my $result = pop_clj $x
# or in a pipeline:
@list |> map pop_clj |> p
```

popc

`popc` — algebraic match builtin. Alias for `pop_clj`.

```
my $result = popc $x
# or in a pipeline:
@list |> map popc |> p
```

popcnt

popcnt — batch 10 builtin. Alias for hamming_weight.

```
my $result = popcnt $x
# or in a pipeline:
@list |> map popcnt |> p
```

popcount

popcount — base conversion builtin. Alias for bits_count.

```
my $result = popcount $x
# or in a pipeline:
@list |> map popcount |> p
```

porosity_active_layer

porosity_active_layer — batch 38 builtin.

```
my $result = porosity_active_layer $x
# or in a pipeline:
@list |> map porosity_active_layer |> p
```

porter_stem_step

porter_stem_step — batch 69 builtin.

```
my $result = porter_stem_step $x
# or in a pipeline:
@list |> map porter_stem_step |> p
```

pos_n

pos_n — trivial numeric / predicate builtins builtin. Alias for positive.

```
my $result = pos_n $x
# or in a pipeline:
@list |> map pos_n |> p
```

poset_zeta_two

poset_zeta_two — batch 37 builtin.

```
my $result = poset_zeta_two $x
# or in a pipeline:
@list |> map poset_zeta_two |> p
```

position_of_all_matching

position_of_all_matching — batch 9 builtin.

```
my $result = position_of_all_matching $x
# or in a pipeline:
@list |> map position_of_all_matching |> p
```

positions_of

`positions_of` — collection (batch 2) builtin.

```
my $result = positions_of $x
# or in a pipeline:
@list |> map positions_of |> p
```

positions_of_all

`positions_of_all` — batch 9 builtin. Alias for `position_of_all_matching`.

```
my $result = positions_of_all $x
# or in a pipeline:
@list |> map positions_of_all |> p
```

positive

`positive` — trivial numeric / predicate builtins builtin.

```
my $result = positive $x
# or in a pipeline:
@list |> map positive |> p
```

post_newtonian_step

`post_newtonian_step` — batch 39 builtin.

```
my $result = post_newtonian_step $x
# or in a pipeline:
@list |> map post_newtonian_step |> p
```

posted_price_offer_accept

`posted_price_offer_accept` — batch 43 builtin.

```
my $result = posted_price_offer_accept $x
# or in a pipeline:
@list |> map posted_price_offer_accept |> p
```

potential_energy

`potential_energy` — physics formulas builtin.

```
my $result = potential_energy $x
# or in a pipeline:
@list |> map potential_energy |> p
```

potential_temperature_step

`potential_temperature_step` — batch 42 builtin.

```
my $result = potential_temperature_step $x
# or in a pipeline:
@list |> map potential_temperature_step |> p
```

pourbaix_line_value

`pourbaix_line_value` — batch 38 builtin.

```
my $result = pourbaix_line_value $x
# or in a pipeline:
@list |> map pourbaix_line_value |> p
```

pow2

`pow2` — trivial numeric / predicate builtins builtin.

```
my $result = pow2 $x
# or in a pipeline:
@list |> map pow2 |> p
```

power_density_battery

`power_density_battery` — batch 38 builtin.

```
my $result = power_density_battery $x
# or in a pipeline:
@list |> map power_density_battery |> p
```

power_i2r

`power_i2r` — batch 23 builtin.

```
my $result = power_i2r $x
# or in a pipeline:
@list |> map power_i2r |> p
```

power_law_wind_profile

`power_law_wind_profile` — batch 42 builtin.

```
my $result = power_law_wind_profile $x
# or in a pipeline:
@list |> map power_law_wind_profile |> p
```

power_phys

`power_phys` — physics formulas builtin.

```
my $result = power_phys $x
# or in a pipeline:
@list |> map power_phys |> p
```

power_residue_check

`power_residue_check` — batch 35 builtin.

```
my $result = power_residue_check $x
# or in a pipeline:
@list |> map power_residue_check |> p
```

power_set

`power_set` — extended stdlib batch 3 builtin.

```
my $result = power_set $x
# or in a pipeline:
@list |> map power_set |> p
```

power_spectrum

`power_spectrum(\@signal)` (alias `psd`) — computes the power spectral density (magnitude squared of DFT). Returns real values representing power at each frequency bin.

```
my $psd = psd(\@signal)
@$psd |> e p
```

power_spectrum_slope

`power_spectrum_slope` — batch 42 builtin.

```
my $result = power_spectrum_slope $x
# or in a pipeline:
@list |> map power_spectrum_slope |> p
```

power_vi

`power_vi` — batch 23 builtin.

```
my $result = power_vi $x
# or in a pipeline:
@list |> map power_vi |> p
```

powers_of_seq

`powers_of_seq` — sequences builtin.

```
my $result = powers_of_seq $x
# or in a pipeline:
@list |> map powers_of_seq |> p
```

powerset

`powerset` — extended stdlib batch 3 builtin. Alias for `power_set`.

```
my $result = powerset $x
# or in a pipeline:
@list |> map powerset |> p
```

powspec

`powspec` — dsp / signal (extended) builtin. Alias for `power_spectrum`.

```
my $result = powspec $x
# or in a pipeline:
@list |> map powspec |> p
```

poynting_magnitude

`poynting_magnitude` — batch 23 builtin.

```
my $result = poynting_magnitude $x
# or in a pipeline:
@list |> map poynting_magnitude |> p
```

pp_test_stat

`pp_test_stat` — batch 36 builtin.

```
my $result = pp_test_stat $x
# or in a pipeline:
@list |> map pp_test_stat |> p
```

ppi

`ppi` — number theory / primes builtin. Alias for `prime_pi`.

```
my $result = ppi $x
# or in a pipeline:
@list |> map ppi |> p
```

ppid

`ppid` — system introspection builtin.

```
my $result = ppid $x
# or in a pipeline:
@list |> map ppid |> p
```

ppm_predict_prob

`ppm_predict_prob` — batch 40 builtin.

```
my $result = ppm_predict_prob $x
# or in a pipeline:
@list |> map ppm_predict_prob |> p
```

ppmt

`ppmt` — batch 57 builtin.

```
my $result = ppmt $x
# or in a pipeline:
@list |> map ppmt |> p
```

ppo_advantage_step

`ppo_advantage_step` — batch 40 builtin.

```
my $result = ppo_advantage_step $x
# or in a pipeline:
@list |> map ppo_advantage_step |> p
```

ppo_clip_term

`ppo_clip_term` — batch 40 builtin.

```
my $result = ppo_clip_term $x
# or in a pipeline:
@list |> map ppo_clip_term |> p
```

ppois

`ppois` — Poisson CDF. P(X □ k) for Poisson(lambda). Args: k, lambda.

```
p ppois(3, 2.5)  # P(≤3 events when avg is 2.5)
```

ppt

`ppt` — extended stdlib builtin. Alias for `partition_point`.

```
my $result = ppt $x
# or in a pipeline:
@list |> map ppt |> p
```

prandtl_number_step

`prandtl_number_step` — batch 42 builtin.

```
my $result = prandtl_number_step $x
# or in a pipeline:
@list |> map prandtl_number_step |> p
```

pratt_parse_step

`pratt_parse_step` — batch 68 builtin.

```
my $result = pratt_parse_step $x
# or in a pipeline:
@list |> map pratt_parse_step |> p
```

precession_iau2006

`precession_iau2006` — batch 55 builtin.

```
my $result = precession_iau2006 $x
# or in a pipeline:
@list |> map precession_iau2006 |> p
```

prect

`prect` — geometry / physics builtin. Alias for `perimeter_rectangle`.

```
my $result = prect $x
# or in a pipeline:
@list |> map prect |> p
```

pred

`pred` — conversion / utility (batch 4) builtin.

```
my $result = pred $x
# or in a pipeline:
@list |> map pred |> p
```

predict

`predict_lm` (alias `predict`) — predict from a linear model at new x values.

```
my $model = lm([1,2,3], [2,4,6])
my @pred = @{predict($model, [4,5,6])}  # [8,10,12]
```

predictor_corrector

`predictor_corrector` — batch 34 builtin.

```
my $result = predictor_corrector $x
# or in a pipeline:
@list |> map predictor_corrector |> p
```

prefix_sums

`prefix_sums` — list helpers (batch 4) builtin.

```
my $result = prefix_sums $x
# or in a pipeline:
@list |> map prefix_sums |> p
```

prelu

`prelu` — batch 27 builtin.

```
my $result = prelu $x
# or in a pipeline:
@list |> map prelu |> p
```

premium_net

`premium_net` — batch 60 builtin.

```
my $result = premium_net $x
# or in a pipeline:
@list |> map premium_net |> p
```

prepend

`prepend` — list helpers (batch 4) builtin.

```
my $result = prepend $x
# or in a pipeline:
@list |> map prepend |> p
```

present_value

`present_value` — physics formulas builtin.

```
my $result = present_value $x
# or in a pipeline:
@list |> map present_value |> p
```

pressure_altitude_m

`pressure_altitude_m` — batch 12 builtin.

```
my $result = pressure_altitude_m $x
# or in a pipeline:
@list |> map pressure_altitude_m |> p
```

prev_prime

`prev_prime` — number theory / primes builtin.

```
my $result = prev_prime $x
# or in a pipeline:
@list |> map prev_prime |> p
```

preventive_fraction

`preventive_fraction` — batch 61 builtin.

```
my $result = preventive_fraction $x
# or in a pipeline:
@list |> map preventive_fraction |> p
```

prewitt_kernel_value

`prewitt_kernel_value` — batch 40 builtin.

```
my $result = prewitt_kernel_value $x
# or in a pipeline:
@list |> map prewitt_kernel_value |> p
```

price_disc

`price_disc` — batch 81 builtin.

```
my $result = price_disc $x
# or in a pipeline:
@list |> map price_disc |> p
```

price_elasticity

`price_elasticity` — batch 73 builtin.

```
my $result = price_elasticity $x
# or in a pipeline:
@list |> map price_elasticity |> p
```

prim_step

`prim_step` — batch 75 builtin.

```
my $result = prim_step $x
# or in a pipeline:
@list |> map prim_step |> p
```

prime_factors

`prime_factors` — number theory / primes builtin.

```
my $result = prime_factors $x
# or in a pipeline:
@list |> map prime_factors |> p
```

prime_pi

`prime_pi` — number theory / primes builtin.

```
my $result = prime_pi $x
# or in a pipeline:
@list |> map prime_pi |> p
```

primes_seq

`primes_seq` — sequences builtin.

```
my $result = primes_seq $x
# or in a pipeline:
@list |> map primes_seq |> p
```

primes_up_to

`primes_up_to` — number theory / primes builtin.

```
my $result = primes_up_to $x
# or in a pipeline:
@list |> map primes_up_to |> p
```

primorial_n

`primorial_n` — batch 35 builtin.

```
my $result = primorial_n $x
# or in a pipeline:
@list |> map primorial_n |> p
```

prisoners_dilemma_payoff

`prisoners_dilemma_payoff` — batch 43 builtin.

```
my $result = prisoners_dilemma_payoff $x
# or in a pipeline:
@list |> map prisoners_dilemma_payoff |> p
```

prisoners_repeated_eq

`prisoners_repeated_eq` — batch 43 builtin.

```
my $result = prisoners_repeated_eq $x
# or in a pipeline:
@list |> map prisoners_repeated_eq |> p
```

priv

`priv` marks a field or method as private. Private members can only be accessed from within the class.

```
class Secret {
    priv hidden: Int = 42
    priv fn internal { }
    pub fn reveal { $self->hidden }  # internal access ok
}
# $obj->hidden  # ERROR: field is private
```

prm_node_connect

`prm_node_connect` — batch 59 builtin.

```
my $result = prm_node_connect $x
# or in a pipeline:
@list |> map prm_node_connect |> p
```

probability_density

`probability_density` — batch 32 builtin.

```
my $result = probability_density $x
# or in a pipeline:
@list |> map probability_density |> p
```

probit_log_likelihood

`probit_log_likelihood` — batch 36 builtin.

```
my $result = probit_log_likelihood $x
# or in a pipeline:
@list |> map probit_log_likelihood |> p
```

proc_mem

`proc_mem` (alias `rss`) — current process resident set size (RSS) in bytes. On Linux reads `VmRSS` from `/proc/self/status`. On macOS uses `getrusage(RUSAGE_SELF)`. Returns `undef` on unsupported platforms.

```
p format_bytes(rss)                    # 18.5 MiB
my $before = rss
do_work()
p format_bytes(rss - $before) . " allocated"
```

producer_surplus

`producer_surplus` — batch 73 builtin.

```
my $result = producer_surplus $x
# or in a pipeline:
@list |> map producer_surplus |> p
```

product_list

`product_list` — list helpers (batch 4) builtin.

```
my $result = product_list $x
# or in a pipeline:
@list |> map product_list |> p
```

product_of

`product_of` — more list helpers builtin.

```
my $result = product_of $x
# or in a pipeline:
@list |> map product_of |> p
```

proj

`proj` — batch 11 builtin. Alias for `vector_project`.

```
my $result = proj $x
# or in a pipeline:
@list |> map proj |> p
```

projectile_max_height

`projectile_max_height($velocity, $angle)` (alias `projheight`) — maximum height $H = v^2 \sin^2(\theta)/(2g)$.

```
p projheight(100, 45)   # ~255 m
p projheight(100, 90)   # ~510 m (straight up)
```

projectile_range

`projectile_range($velocity, $angle)` (alias `projrange`) — horizontal range $R = v^2 \sin(2\theta)/g$. Angle in degrees.

```
p projrange(100, 45)   # ~1020 m (optimal angle for range)
p projrange(100, 30)   # ~884 m
```

projectile_time

`projectile_time` — physics formulas builtin.

```
my $result = projectile_time $x
# or in a pipeline:
@list |> map projectile_time |> p
```

projtime

`projtime` — physics formulas builtin. Alias for `projectile_time`.

```
my $result = projtime $x
# or in a pipeline:
@list |> map projtime |> p
```

pronic_number

`pronic_number` — batch 11 builtin.

```
my $result = pronic_number $x
# or in a pipeline:
@list |> map pronic_number |> p
```

prop_table

`prop_table` — convert a count matrix to proportions (each cell / total). Like R's prop.table().

```
p prop_table([[10,20],[30,40]])  # [[0.1,0.2],[0.3,0.4]]
```

prop_test

`prop_test` — one-sample proportion z-test. Returns [z-statistic, p-value]. Like R's prop.test().

```
my ($z, $pval) = @{proptest(55, 100, 0.5)}  # test if 55/100 differs from 50%%
```

proper_motion_apply

`proper_motion_apply` — batch 55 builtin.

```
my $result = proper_motion_apply $x
# or in a pipeline:
@list |> map proper_motion_apply |> p
```

proportional_share

`proportional_share` — batch 43 builtin.

```
my $result = proportional_share $x
# or in a pipeline:
@list |> map proportional_share |> p
```

proton_mass

`proton_mass` — constants builtin.

```
my $result = proton_mass $x
# or in a pipeline:
@list |> map proton_mass |> p
```

proximal_gradient_step

`proximal_gradient_step` — batch 41 builtin.

```
my $result = proximal_gradient_step $x
# or in a pipeline:
@list |> map proximal_gradient_step |> p
```

prvp

`prvp` — number theory / primes builtin. Alias for `prev_prime`.

```
my $result = prvp $x
# or in a pipeline:
@list |> map prvp |> p
```

pseudoprime_base2

`pseudoprime_base2` — batch 35 builtin.

```
my $result = pseudoprime_base2 $x
# or in a pipeline:
@list |> map pseudoprime_base2 |> p
```

psi_to_pascals

`psi_to_pascals` — file stat / path builtin.

```
my $result = psi_to_pascals $input
```

pt

`pt` — Student's t CDF. Args: x, df.

```
p pt(1.96, 100)  # ≈ 0.975
```

ptdist

`ptdist` — geometry / physics builtin. Alias for `point_distance`.

```
my $result = ptdist $x
# or in a pipeline:
@list |> map ptdist |> p
```

ptinpoly

`ptinpoly` — geometry (extended) builtin. Alias for `point_in_polygon`.

```
my $result = ptinpoly $x
# or in a pipeline:
@list |> map ptinpoly |> p
```

ptri

`ptri` — geometry / physics builtin. Alias for `perimeter_triangle`.

```
my $result = ptri $x
# or in a pipeline:
@list |> map ptri |> p
```

pttl

`pttl` — batch 71 builtin.

```
my $result = pttl $x
# or in a pipeline:
@list |> map pttl |> p
```

pub

`pub` marks a field or method as public (default visibility).

```
class Example {
    pub name: Str       # explicitly public
    pub fn greet { }    # explicitly public
}
```

public_goods_game_payoff

`public_goods_game_payoff` — batch 43 builtin.

```
my $result = public_goods_game_payoff $x
# or in a pipeline:
@list |> map public_goods_game_payoff |> p
```

pulse_pressure

`pulse_pressure` — batch 12 builtin.

```
my $result = pulse_pressure $x
# or in a pipeline:
@list |> map pulse_pressure |> p
```

punct

`punctuation` (alias `punct`) extracts all ASCII punctuation characters from a string and returns them as a list. Filters out letters, digits, whitespace — keeps only punctuation.

```
p join "", punctuation("Hello, world!")  # ,!
"a.b-c" |> punct |> cnt |> p             # 2
```

punif

`punif` — uniform CDF. P(X ≤ x) for Uniform(a, b). Args: x, a, b.

```
p punif(0.5, 0, 1)  # 0.5
p punif(3, 0, 10)   # 0.3
```

punycode_decode_step

`punycode_decode_step` — batch 81 builtin.

```
my $result = punycode_decode_step $x
# or in a pipeline:
@list |> map punycode_decode_step |> p
```

punycode_encode

`punycode_encode` — batch 62 builtin.

```
my $result = punycode_encode $x
# or in a pipeline:
@list |> map punycode_encode |> p
```

pure_state_density

`pure_state_density` — batch 32 builtin.

```
my $result = pure_state_density $x
# or in a pipeline:
@list |> map pure_state_density |> p
```

purity

`purity` — batch 32 builtin.

```
my $result = purity $x
# or in a pipeline:
@list |> map purity |> p
```

purple

`purple` — color / ansi builtin. Alias for `red`.

```
my $result = purple $x
# or in a pipeline:
@list |> map purple |> p
```

push_relabel_relabel

`push_relabel_relabel` — batch 75 builtin.

```
my $result = push_relabel_relabel $x
# or in a pipeline:
@list |> map push_relabel_relabel |> p
```

push_relabel_step

`push_relabel_step` — batch 41 builtin.

```
my $result = push_relabel_step $x
# or in a pipeline:
@list |> map push_relabel_step |> p
```

pv

`pv` — finance (extended) builtin.

```
my $result = pv $x
# or in a pipeline:
@list |> map pv |> p
```

pwave_velocity_depth

`pwave_velocity_depth` — batch 65 builtin.

```
my $result = pwave_velocity_depth $x
# or in a pipeline:
@list |> map pwave_velocity_depth |> p
```

pwd_str

`pwd_str` — more process / system builtin. Alias for `cwd`.

```
my $result = pwd_str $x
# or in a pipeline:
@list |> map pwd_str |> p
```

pweibull

`pweibull` — Weibull CDF. P(X ≤ x) for Weibull(shape, scale). Args: x, shape, scale.

```
p pweibull(1, 1, 1)  # same as exponential
```

pwi

`pwi` — python/ruby stdlib builtin. Alias for `pairwise_iter`.

```
my $result = pwi $x
# or in a pipeline:
@list |> map pwi |> p
```

pwm_score

`pwm_score` — batch 33 builtin.

```
my $result = pwm_score $x
# or in a pipeline:
@list |> map pwm_score |> p
```

pyramid_volume

`pyramid_volume($base_area, $height)` — computes the volume of a pyramid. Returns `(1/3) * base_area * height`.

```
p pyramid_volume(100, 15)   # 500
```

pyrvol

`pyrvol` — geometry (extended) builtin. Alias for `pyramid_volume`.

```
my $result = pyrvol $x
# or in a pipeline:
@list |> map pyrvol |> p
```

pythagorean_expectation

`pythagorean_expectation` — batch 56 builtin.

```
my $result = pythagorean_expectation $x
# or in a pipeline:
@list |> map pythagorean_expectation |> p
```

pythagorean_freq

`pythagorean_freq` — batch 64 builtin.

```
my $result = pythagorean_freq $x
# or in a pipeline:
@list |> map pythagorean_freq |> p
```

pythagorean_ratio

`pythagorean_ratio` — batch 30 builtin.

```
my $result = pythagorean_ratio $x
# or in a pipeline:
@list |> map pythagorean_ratio |> p
```

q

`q` — quoting builtin.

```
my $result = q $x
# or in a pipeline:
@list |> map q |> p
```

q10

`q10` — batch 22 builtin.

```
my $result = q10 $x
# or in a pipeline:
@list |> map q10 |> p
```

q_factor_rlc

`q_factor_rlc` — batch 12 builtin.

```
my $result = q_factor_rlc $x
# or in a pipeline:
@list |> map q_factor_rlc |> p
```

q_learning_update

`q_learning_update` — batch 40 builtin.

```
my $result = q_learning_update $x
# or in a pipeline:
@list |> map q_learning_update |> p
```

qaly_lifetime

`qaly_lifetime` — batch 61 builtin.

```
my $result = qaly_lifetime $x
# or in a pipeline:
@list |> map qaly_lifetime |> p
```

qaoa_step

`qaoa_step` — batch 80 builtin.

```
my $result = qaoa_step $x
# or in a pipeline:
@list |> map qaoa_step |> p
```

qap_lower_bound

`qap_lower_bound` — batch 41 builtin.

```
my $result = qap_lower_bound $x
# or in a pipeline:
@list |> map qap_lower_bound |> p
```

qbeta

`qbeta` — beta quantile. Args: p, alpha, beta.

```
p qbeta(0.5, 2, 5)  # median of Beta(2,5)
```

qbinom

`qbinom` — binomial quantile (smallest k where P(X□k) □ p). Args: p, n, prob.

```
p qbinom(0.5, 10, 0.5)  # median of Binom(10, 0.5) = 5
```

qbo_oscillation_step

`qbo_oscillation_step` — batch 42 builtin.

```
my $result = qbo_oscillation_step $x
# or in a pipeline:
@list |> map qbo_oscillation_step |> p
```

qbr_metric

`qbr_metric` — batch 56 builtin.

```
my $result = qbr_metric $x
# or in a pipeline:
@list |> map qbr_metric |> p
```

qcauchy

`qcauchy` — Cauchy quantile. Args: p [, location, scale].

```
p qcauchy(0.75, 0, 1)  # 1.0
```

qchisq

`qchisq` — chi-squared quantile. Args: p, df.

```
p qchisq(0.95, 1)  # 3.84 (critical value)
p qchisq(0.95, 5)  # 11.07
```

qdisc

`quadratic_discriminant($a, $b, $c)` (alias `qdisc`) — computes $b^2 - 4ac$. Positive = two real roots, zero = one repeated root, negative = complex roots.

```
p qdisc(1, -5, 6)   # 1 (two distinct real roots)
p qdisc(1, 2, 1)    # 0 (one repeated root)
p qdisc(1, 2, 5)    # -16 (complex roots)
```

qexp

`qexp` — exponential quantile. Args: p [, rate].

```
p qexp(0.5, 1)       # 0.693 (median of Exp(1))
```

qf

`qf` — F-distribution quantile. Args: p, d1, d2.

```
p qf(0.95, 5, 10)  # critical F value
```

qft

`qft` — batch 80 builtin.

```
my $result = qft $x
# or in a pipeline:
@list |> map qft |> p
```

qft_4_real

`qft_4_real` — batches 14-16 builtin.

```
my $result = qft_4_real $x
# or in a pipeline:
@list |> map qft_4_real |> p
```

qgamma

`qgamma` — gamma quantile (inverse CDF). Args: p, shape [, scale].

```
p qgamma(0.95, 2, 1)  # 95th percentile of Gamma(2,1)
```

qho_ground_state

`qho_ground_state` — batch 32 builtin.

```
my $result = qho_ground_state $x
# or in a pipeline:
@list |> map qho_ground_state |> p
```

qlnorm

`qlnorm` — log-normal quantile. Args: p [, mu, sigma].

```
p qlnorm(0.5, 0, 1)  # 1.0 (median of LogN(0,1))
```

qm

`qm` — string builtin. Alias for `quotemeta`.

```
my $result = qm $x
# or in a pipeline:
@list |> map qm |> p
```

qnorm

`qnorm` — normal quantile (inverse CDF). Returns x such that P(X ≤ x) = p. Args: p [, mu, sigma].

```
p qnorm(0.975)      # 1.96
p qnorm(0.5)        # 0.0
p qnorm(0.95, 100, 15)  # 90th percentile IQ
```

qntl

`qntl` — extended stdlib builtin. Alias for `quantile`.

```
my $result = qntl $x
# or in a pipeline:
@list |> map qntl |> p
```

qntls

`quantiles \@DATA, \@PROBS` — batch quantile lookup. Returns one element per probability in `\@PROBS`, computed via the same linear interpolation as `quantile`. Sorts the data exactly once across all probability points so callers don't pay an O(n log n) sort per quantile. Probabilities outside [0,1] are clamped.

```
my @d = (1..1000)
my @qs = quantiles(\@d, [0.25, 0.5, 0.75])
printf "q25=%.2f q50=%.2f q75=%.2f\n", @qs
# q25=250.75 q50=500.50 q75=750.25
```

qpe_iteration

`qpe_iteration` — batch 80 builtin.

```
my $result = qpe_iteration $x
# or in a pipeline:
@list |> map qpe_iteration |> p
```

qpois

`qpois` — Poisson quantile (smallest k where P(X□k) □ p). Args: p, lambda.

```
p qpois(0.5, 5)  # median of Poisson(5)
```

qq

`qq` — quoting builtin.

```
my $result = qq $x
# or in a pipeline:
@list |> map qq |> p
```

qroots

`quadratic_roots($a, $b, $c)` (alias `qroots`) — solves $ax^2 + bx + c = 0$. Returns [x1, x2] for real roots, undef if no real solutions (discriminant < 0).

```
my @roots = quadratic_roots(1, -5, 6)  # [3, 2] (x² - 5x + 6 = 0)
p qroots(1, 0, -4)                     # [2, -2]
p qroots(1, 2, 5)                      # undef (complex roots)
```

qs_relation

`qs_relation` — batch 35 builtin.

```
my $result = qs_relation $x
# or in a pipeline:
@list |> map qs_relation |> p
```

qt

`qt` — Student's t quantile. Args: p, df.

```
p qt(0.975, 10)   # ≈ 2.228 (two-tailed 5%)
```

quadratic_assignment_step

quadratic_assignment_step — batch 41 builtin.

```
my $result = quadratic_assignment_step $x
# or in a pipeline:
@list |> map quadratic_assignment_step |> p
```

quadratic_mean_arr

quadratic_mean_arr — batch 11 builtin.

```
my $result = quadratic_mean_arr $x
# or in a pipeline:
@list |> map quadratic_mean_arr |> p
```

quantal_response_logit

quantal_response_logit — batch 43 builtin.

```
my $result = quantal_response_logit $x
# or in a pipeline:
@list |> map quantal_response_logit |> p
```

quantile

quantile — extended stdlib builtin.

```
my $result = quantile $x
# or in a pipeline:
@list |> map quantile |> p
```

quantile_p

quantile_p — batches 14-16 builtin.

```
my $result = quantile_p $x
# or in a pipeline:
@list |> map quantile_p |> p
```

quantize

quantize — batch 25 builtin.

```
my $result = quantize $x
# or in a pipeline:
@list |> map quantize |> p
```

quanto_adjustment

quanto_adjustment — batch 20 builtin.

```
my $result = quanto_adjustment $input
```

quantum_dimension_q

quantum_dimension_q — batch 37 builtin.

```
my $result = quantum_dimension_q $x
# or in a pipeline:
@list |> map quantum_dimension_q |> p
```

quantum_dimension_su2

quantum_dimension_su2 — batch 37 builtin.

```
my $result = quantum_dimension_su2 $x
# or in a pipeline:
@list |> map quantum_dimension_su2 |> p
```

quantum_efficiency_photo

quantum_efficiency_photo — batch 38 builtin.

```
my $result = quantum_efficiency_photo $x
# or in a pipeline:
@list |> map quantum_efficiency_photo |> p
```

quantum_fidelity_pure

quantum_fidelity_pure — batches 14-16 builtin.

```
my $result = quantum_fidelity_pure $x
# or in a pipeline:
@list |> map quantum_fidelity_pure |> p
```

quantum_fisher_info

quantum_fisher_info — batch 32 builtin.

```
my $result = quantum_fisher_info $x
# or in a pipeline:
@list |> map quantum_fisher_info |> p
```

quantum_invariant_eval

quantum_invariant_eval — batch 37 builtin.

```
my $result = quantum_invariant_eval $x
# or in a pipeline:
@list |> map quantum_invariant_eval |> p
```

quantum_mutual_info

quantum_mutual_info — batch 32 builtin.

```
my $result = quantum_mutual_info $x
# or in a pipeline:
@list |> map quantum_mutual_info |> p
```

quantum_relative_entropy

quantum_relative_entropy — batches 14-16 builtin.

```
my $result = quantum_relative_entropy $x
# or in a pipeline:
@list |> map quantum_relative_entropy |> p
```

quantum_teleportation

`quantum_teleportation` — batch 80 builtin.

```
my $result = quantum_teleportation $x
# or in a pipeline:
@list |> map quantum_teleportation |> p
```

quantum_variance

`quantum_variance` — batch 32 builtin.

```
my $result = quantum_variance $x
# or in a pipeline:
@list |> map quantum_variance |> p
```

quarter_of

`quarter_of` — date helpers builtin.

```
my $result = quarter_of $x
# or in a pipeline:
@list |> map quarter_of |> p
```

quarter_of_year

`quarter_of_year` — batch 13 builtin.

```
my $result = quarter_of_year $x
# or in a pipeline:
@list |> map quarter_of_year |> p
```

quartile_exc

`quartile_exc` — batch 57 builtin.

```
my $result = quartile_exc $x
# or in a pipeline:
@list |> map quartile_exc |> p
```

quartile_inc

`quartile_inc` — batch 57 builtin.

```
my $result = quartile_inc $x
# or in a pipeline:
@list |> map quartile_inc |> p
```

quartiles

`quartiles(@data)` — returns the three quartile values [Q1, Q2, Q3] of a dataset. Q1 is the 25th percentile, Q2 is the median (50th), Q3 is the 75th percentile. Useful for understanding data distribution and computing IQR.

```
my @data = 1:100
my $q = quartiles(@data)
p "Q1=$q->[0] Q2=$q->[1] Q3=$q->[2]"
```

quarts

`quarts` — statistics (extended) builtin. Alias for `quartiles`.

```
my $result = quarts $x
# or in a pipeline:
@list |> map quarts |> p
```

quasi_geostrophic_omega

`quasi_geostrophic_omega` — batch 42 builtin.

```
my $result = quasi_geostrophic_omega $x
# or in a pipeline:
@list |> map quasi_geostrophic_omega |> p
```

quaternion_from_imu

`quaternion_from_imu` — batch 59 builtin.

```
my $result = quaternion_from_imu $x
# or in a pipeline:
@list |> map quaternion_from_imu |> p
```

qubit_ccx

`qubit_ccx` — batch 80 builtin.

```
my $result = qubit_ccx $x
# or in a pipeline:
@list |> map qubit_ccx |> p
```

qubit_cnot

`qubit_cnot` — batch 80 builtin.

```
my $result = qubit_cnot $x
# or in a pipeline:
@list |> map qubit_cnot |> p
```

qubit_cz

`qubit_cz` — batch 80 builtin.

```
my $result = qubit_cz $x
# or in a pipeline:
@list |> map qubit_cz |> p
```

qubit_h

`qubit_h` — batch 80 builtin.

```
my $result = qubit_h $x
# or in a pipeline:
@list |> map qubit_h |> p
```

qubit_measure

`qubit_measure` — batch 80 builtin.

```
my $result = qubit_measure $x
# or in a pipeline:
@list |> map qubit_measure |> p
```

qubit_phase

qubit_phase — batch 80 builtin.

```
my $result = qubit_phase $x
# or in a pipeline:
@list |> map qubit_phase |> p
```

qubit_reset

qubit_reset — batch 80 builtin.

```
my $result = qubit_reset $x
# or in a pipeline:
@list |> map qubit_reset |> p
```

qubit_rx

qubit_rx — batch 80 builtin.

```
my $result = qubit_rx $x
# or in a pipeline:
@list |> map qubit_rx |> p
```

qubit_ry

qubit_ry — batch 80 builtin.

```
my $result = qubit_ry $x
# or in a pipeline:
@list |> map qubit_ry |> p
```

qubit_rz

qubit_rz — batch 80 builtin.

```
my $result = qubit_rz $x
# or in a pipeline:
@list |> map qubit_rz |> p
```

qubit_s

qubit_s — batch 80 builtin.

```
my $result = qubit_s $x
# or in a pipeline:
@list |> map qubit_s |> p
```

qubit_swap

qubit_swap — batch 80 builtin.

```
my $result = qubit_swap $x
# or in a pipeline:
@list |> map qubit_swap |> p
```

qubit_t

qubit_t — batch 80 builtin.

```
my $result = qubit_t $x
# or in a pipeline:
@list |> map qubit_t |> p
```

qubit_u1

qubit_u1 — batch 80 builtin.

```
my $result = qubit_u1 $x
# or in a pipeline:
@list |> map qubit_u1 |> p
```

qubit_u2

qubit_u2 — batch 80 builtin.

```
my $result = qubit_u2 $x
# or in a pipeline:
@list |> map qubit_u2 |> p
```

qubit_u3

qubit_u3 — batch 80 builtin.

```
my $result = qubit_u3 $x
# or in a pipeline:
@list |> map qubit_u3 |> p
```

qubit_x

qubit_x — batch 80 builtin.

```
my $result = qubit_x $x
# or in a pipeline:
@list |> map qubit_x |> p
```

qubit_y

qubit_y — batch 80 builtin.

```
my $result = qubit_y $x
# or in a pipeline:
@list |> map qubit_y |> p
```

qubit_z

qubit_z — batch 80 builtin.

```
my $result = qubit_z $x
# or in a pipeline:
@list |> map qubit_z |> p
```

queue_new

queue_new — data structure helpers builtin.

```
my $result = queue_new $x
# or in a pipeline:
@list |> map queue_new |> p
```

quiver_path_count

quiver_path_count — batch 37 builtin.

```
my $result = quiver_path_count $x
# or in a pipeline:
@list |> map quiver_path_count |> p
```

qunif

`qunif` — uniform quantile. Args: p [, min, max].

```
p qunif(0.5, 0, 10)  # 5.0 (median)
```

quote

`quote` — string quote / escape builtin.

```
my $result = quote $x
# or in a pipeline:
@list |> map quote |> p
```

quoted_printable_encode

`quoted_printable_encode` — batch 62 builtin.

```
my $result = quoted_printable_encode $x
# or in a pipeline:
@list |> map quoted_printable_encode |> p
```

qw

`qw` — quoting builtin.

```
my $result = qw $x
# or in a pipeline:
@list |> map qw |> p
```

qweibull

`qweibull` — Weibull quantile. Args: p, shape [, scale].

```
p qweibull(0.5, 1, 1)  # same as qexp
```

r0_basic

`r0_basic` — batch 22 builtin.

```
my $result = r0_basic $x
# or in a pipeline:
@list |> map r0_basic |> p
```

r2d

`r2d` — trivial numeric / predicate builtins builtin. Alias for `rad_to_deg`.

```
my $result = r2d $x
# or in a pipeline:
@list |> map r2d |> p
```

r72

`rule_of_72($rate)` (alias `r72`) — estimates years to double investment using the Rule of 72: 72 / (rate × 100). Quick mental math approximation.

```
p r72(0.06)   # 12 years to double at 6%
p r72(0.10)   # 7.2 years at 10%
```

r_effective_t

`r_effective_t` — batch 61 builtin.

```
my $result = r_effective_t $x
# or in a pipeline:
@list |> map r_effective_t |> p
```

r_naught_basic

`r_naught_basic` — batch 61 builtin.

```
my $result = r_naught_basic $x
# or in a pipeline:
@list |> map r_naught_basic |> p
```

r_parallel

`r_parallel` — batch 12 builtin. Alias for `resistance_parallel`.

```
my $result = r_parallel $x
# or in a pipeline:
@list |> map r_parallel |> p
```

r_series

`r_series` — batch 12 builtin. Alias for `resistance_series`.

```
my $result = r_series $x
# or in a pipeline:
@list |> map r_series |> p
```

r_squared

`r_squared` — math functions builtin.

```
my $result = r_squared $x
# or in a pipeline:
@list |> map r_squared |> p
```

ra_dec_to_alt_az

`ra_dec_to_alt_az` — batch 55 builtin.

```
my $result = ra_dec_to_alt_az $input
```

ra_dec_to_az_alt

`ra_dec_to_az_alt` — batches 14-16 builtin.

```
my $result = ra_dec_to_az_alt $input
```

rabi_frequency

`rabi_frequency` — batch 32 builtin.

```
my $result = rabi_frequency $x
# or in a pipeline:
@list |> map rabi_frequency |> p
```

rabin_karp_hash

`rabin_karp_hash` — batch 10 builtin.

```
my $result = rabin_karp_hash $x
# or in a pipeline:
@list |> map rabin_karp_hash |> p
```

rad_to_deg

`rad_to_deg` — trivial numeric / predicate builtins builtin.

```
my $result = rad_to_deg $input
```

radd

`radd` — roman numerals builtin. Alias for `roman_add`.

```
my $result = radd $x
# or in a pipeline:
@list |> map radd |> p
```

radians

`radians` — trig / math (batch 2) builtin.

```
my $result = radians $x
# or in a pipeline:
@list |> map radians |> p
```

radiation_pressure

`radiation_pressure` — batch 23 builtin.

```
my $result = radiation_pressure $x
# or in a pipeline:
@list |> map radiation_pressure |> p
```

radius_bfs

`radius_bfs` — batch 24 builtin.

```
my $result = radius_bfs $x
# or in a pipeline:
@list |> map radius_bfs |> p
```

ragone_point

`ragone_point` — batch 38 builtin.

```
my $result = ragone_point $x
# or in a pipeline:
@list |> map ragone_point |> p
```

railfence_encrypt

railfence_encrypt — batch 26 builtin.

```
my $result = railfence_encrypt $x
# or in a pipeline:
@list |> map railfence_encrypt |> p
```

ramachandran_phi_psi

ramachandran_phi_psi — batch 63 builtin.

```
my $result = ramachandran_phi_psi $x
# or in a pipeline:
@list |> map ramachandran_phi_psi |> p
```

ramsey_reset_test

ramsey_reset_test — batch 36 builtin.

```
my $result = ramsey_reset_test $x
# or in a pipeline:
@list |> map ramsey_reset_test |> p
```

ramsey_visibility

ramsey_visibility — batch 32 builtin.

```
my $result = ramsey_visibility $x
# or in a pipeline:
@list |> map ramsey_visibility |> p
```

randles_circuit_z

randles_circuit_z — batch 38 builtin.

```
my $result = randles_circuit_z $x
# or in a pipeline:
@list |> map randles_circuit_z |> p
```

randles_sevcik_peak

randles_sevcik_peak — batch 38 builtin.

```
my $result = randles_sevcik_peak $x
# or in a pipeline:
@list |> map randles_sevcik_peak |> p
```

random_alpha

random_alpha — random builtin.

```
my $result = random_alpha $x
# or in a pipeline:
@list |> map random_alpha |> p
```

random_between

random_between — random builtin.

```
my $result = random_between $x
# or in a pipeline:
@list |> map random_between |> p
```

random_bool

`random_bool` — random builtin.

```
my $result = random_bool $x
# or in a pipeline:
@list |> map random_bool |> p
```

random_char

`random_char` — file stat / path builtin.

```
my $result = random_char $x
# or in a pipeline:
@list |> map random_char |> p
```

random_choice

`random_choice` — random builtin.

```
my $result = random_choice $x
# or in a pipeline:
@list |> map random_choice |> p
```

random_color

`random_color` — color operations builtin.

```
my $result = random_color $x
# or in a pipeline:
@list |> map random_color |> p
```

random_digit

`random_digit` — random builtin.

```
my $result = random_digit $x
# or in a pipeline:
@list |> map random_digit |> p
```

random_float

`random_float` — random builtin.

```
my $result = random_float $x
# or in a pipeline:
@list |> map random_float |> p
```

random_int

`random_int` — random builtin.

```
my $result = random_int $x
# or in a pipeline:
@list |> map random_int |> p
```

random_sample

`random_sample` — collection (batch 2) builtin.

```
my $result = random_sample $x
# or in a pipeline:
@list |> map random_sample |> p
```

random_string

`random_string` — random builtin.

```
my $result = random_string $x
# or in a pipeline:
@list |> map random_string |> p
```

random_walk_drift_step

`random_walk_drift_step` — batch 36 builtin.

```
my $result = random_walk_drift_step $x
# or in a pipeline:
@list |> map random_walk_drift_step |> p
```

random_walk_hitting

`random_walk_hitting` — batch 75 builtin.

```
my $result = random_walk_hitting $x
# or in a pipeline:
@list |> map random_walk_hitting |> p
```

random_walk_innovation

`random_walk_innovation` — batch 36 builtin.

```
my $result = random_walk_innovation $x
# or in a pipeline:
@list |> map random_walk_innovation |> p
```

range_coding_step

`range_coding_step` — batch 40 builtin.

```
my $result = range_coding_step $x
# or in a pipeline:
@list |> map range_coding_step |> p
```

range_compress

`range_compress` — matrix operations (uncategorized batch) builtin.

```
my $result = range_compress $x
# or in a pipeline:
@list |> map range_compress |> p
```

range_decode_step

`range_decode_step` — batch 40 builtin.

```
my $result = range_decode_step $x
# or in a pipeline:
@list |> map range_decode_step |> p
```

range_exclusive

`range_exclusive` — conversion / utility (batch 4) builtin.

```
my $result = range_exclusive $x
# or in a pipeline:
@list |> map range_exclusive |> p
```

range_expand

`range_expand` — matrix operations (uncategorized batch) builtin.

```
my $result = range_expand $x
# or in a pipeline:
@list |> map range_expand |> p
```

range_header_parse

`range_header_parse` — batch 81 builtin.

```
my $result = range_header_parse $x
# or in a pipeline:
@list |> map range_header_parse |> p
```

range_inclusive

`range_inclusive` — conversion / utility (batch 4) builtin.

```
my $result = range_inclusive $x
# or in a pipeline:
@list |> map range_inclusive |> p
```

range_of

`range_of` — more list helpers builtin.

```
my $result = range_of $x
# or in a pipeline:
@list |> map range_of |> p
```

range_voting_score

`range_voting_score` — batch 43 builtin.

```
my $result = range_voting_score $x
# or in a pipeline:
@list |> map range_voting_score |> p
```

rank

`rank` — extended stdlib builtin.

```
my $result = rank $x
# or in a pipeline:
@list |> map rank |> p
```

rank_avg

`rank_avg` — batch 57 builtin.

```
my $result = rank_avg $x
# or in a pipeline:
@list |> map rank_avg |> p
```

rank_eq

`rank_eq` — batch 57 builtin.

```
my $result = rank_eq $x
# or in a pipeline:
@list |> map rank_eq |> p
```

ransac_homography

`ransac_homography` — batch 77 builtin.

```
my $result = ransac_homography $x
# or in a pipeline:
@list |> map ransac_homography |> p
```

ransac_iteration_count

`ransac_iteration_count` — batch 40 builtin.

```
my $result = ransac_iteration_count $x
# or in a pipeline:
@list |> map ransac_iteration_count |> p
```

rao_quadratic_entropy

`rao_quadratic_entropy` — batch 22 builtin.

```
my $result = rao_quadratic_entropy $x
# or in a pipeline:
@list |> map rao_quadratic_entropy |> p
```

raoult_law

`raoult_law` — batch 21 builtin.

```
my $result = raoult_law $x
# or in a pipeline:
@list |> map raoult_law |> p
```

rapid_blink

`rapid_blink` — color / ansi builtin. Alias for `red`.

```
my $result = rapid_blink $x
# or in a pipeline:
@list |> map rapid_blink |> p
```

rate

`rate` — batch 57 builtin.

```
my $result = rate $x
# or in a pipeline:
@list |> map rate |> p
```

raychaudhuri_step

raychaudhuri_step — batch 39 builtin.

```
my $result = raychaudhuri_step $x
# or in a pipeline:
@list |> map raychaudhuri_step |> p
```

rayleigh_criterion

rayleigh_criterion — batch 55 builtin.

```
my $result = rayleigh_criterion $x
# or in a pipeline:
@list |> map rayleigh_criterion |> p
```

rayleigh_number_step

rayleigh_number_step — batch 42 builtin.

```
my $result = rayleigh_number_step $x
# or in a pipeline:
@list |> map rayleigh_number_step |> p
```

rayleigh_resolution

rayleigh_resolution — batch 23 builtin.

```
my $result = rayleigh_resolution $x
# or in a pipeline:
@list |> map rayleigh_resolution |> p
```

rb

rb — filesystem extensions builtin. Alias for read_bytes.

```
my $result = rb $x
# or in a pipeline:
@list |> map rb |> p
```

rbeta

rbeta — generate n random beta variates. Args: n, alpha, beta. Like R's rbeta().

```
my @x = @{rbeta(100, 2, 5)}     # Beta(2,5)
```

rbind

rbind — bind matrices/vectors by rows (vertical stack). Like R's rbind().

```
my $m = rbind([1,2,3], [4,5,6])  # [[1,2,3],[4,5,6]]
```

rbinom

rbinom — generate n random binomial variates. Args: n, size, prob. Like R's rbinom().

```
my @x = @{rbinom(100, 10, 0.3)} # 100 draws from Binom(10,0.3)
```

rc_tau

`rc_tau` — batch 23 builtin.

```
my $result = rc_tau $x
# or in a pipeline:
@list |> map rc_tau |> p
```

rc_time_constant

`rc_time_constant` — physics formulas builtin.

```
my $result = rc_time_constant $x
# or in a pipeline:
@list |> map rc_time_constant |> p
```

rcauchy

`rcauchy` — generate n random Cauchy variates. Args: n [, location, scale].

```
my @x = @{rcauchy(100, 0, 1)}   # standard Cauchy
```

rchisq

`rchisq` — generate n random chi-squared variates. Args: n, df. Like R's rchisq().

```
my @x = @{rchisq(100, 5)}       # Chi-sq with 5 df
```

rcolor

`rcolor` — color operations builtin. Alias for `random_color`.

```
my $result = rcolor $x
# or in a pipeline:
@list |> map rcolor |> p
```

rctau

`rctau` — physics formulas builtin. Alias for `rc_time_constant`.

```
my $result = rctau $x
# or in a pipeline:
@list |> map rctau |> p
```

rdcs

`rdcs` — algebraic match builtin. Alias for `reductions`.

```
my $result = rdcs $x
# or in a pipeline:
@list |> map rdcs |> p
```

rdd_estimate

`rdd_estimate` — batch 73 builtin.

```
my $result = rdd_estimate $x
# or in a pipeline:
@list |> map rdd_estimate |> p
```

re_escape

`re_escape` — more regex builtin.

```
my $result = re_escape $x
# or in a pipeline:
@list |> map re_escape |> p
```

re_find_all

`re_find_all` — more regex builtin.

```
my $result = re_find_all $x
# or in a pipeline:
@list |> map re_find_all |> p
```

re_groups

`re_groups` — more regex builtin.

```
my $result = re_groups $x
# or in a pipeline:
@list |> map re_groups |> p
```

re_match

`re_match` — file stat / path builtin. Alias for `matches_regex`.

```
my $result = re_match $x
# or in a pipeline:
@list |> map re_match |> p
```

re_split_limit

`re_split_limit` — more regex builtin.

```
my $result = re_split_limit $x
# or in a pipeline:
@list |> map re_split_limit |> p
```

re_test

`re_test` — more regex builtin.

```
my $result = re_test $x
# or in a pipeline:
@list |> map re_test |> p
```

reaction_quotient

`reaction_quotient` — batch 21 builtin.

```
my $result = reaction_quotient $x
# or in a pipeline:
@list |> map reaction_quotient |> p
```

realized_volatility

`realized_volatility` — batches 17-18 builtin.

```
my $result = realized_volatility $x
# or in a pipeline:
@list |> map realized_volatility |> p
```

rearth

`rearth` — physics constants builtin. Alias for `earth_radius`.

```
my $result = rearth $x
# or in a pipeline:
@list |> map rearth |> p
```

received

`received` — batch 81 builtin.

```
my $result = received $x
# or in a pipeline:
@list |> map received |> p
```

recip

`recip` — extended stdlib builtin. Alias for `reciprocal_of`.

```
my $result = recip $x
# or in a pipeline:
@list |> map recip |> p
```

reciprocal

`reciprocal` — math functions builtin.

```
my $result = reciprocal $x
# or in a pipeline:
@list |> map reciprocal |> p
```

reciprocal_of

`reciprocal_of` — extended stdlib builtin.

```
my $result = reciprocal_of $x
# or in a pipeline:
@list |> map reciprocal_of |> p
```

reciprocal_pdf

`reciprocal_pdf` — batches 17-18 builtin.

```
my $result = reciprocal_pdf $x
# or in a pipeline:
@list |> map reciprocal_pdf |> p
```

red

`red` — color / ansi builtin.

```
my $result = red $x
# or in a pipeline:
@list |> map red |> p
```

red_bold

`red_bold` — color / ansi builtin. Alias for `red`.

```
my $result = red_bold $x
# or in a pipeline:
@list |> map red_bold |> p
```

redi

`redi` — go/general functional utilities builtin. Alias for `reduce_indexed`.

```
my $result = redi $x
# or in a pipeline:
@list |> map redi |> p
```

redlich_kwong_p

`redlich_kwong_p` — batch 21 builtin.

```
my $result = redlich_kwong_p $x
# or in a pipeline:
@list |> map redlich_kwong_p |> p
```

redox_potential_cell

`redox_potential_cell` — batch 63 builtin.

```
my $result = redox_potential_cell $x
# or in a pipeline:
@list |> map redox_potential_cell |> p
```

redr

`redr` — additional missing stdlib functions builtin. Alias for `reduce_right`.

```
my $result = redr $x
# or in a pipeline:
@list |> map redr |> p
```

redshift_from_a

`redshift_from_a` — batch 31 builtin.

```
my $result = redshift_from_a $x
# or in a pipeline:
@list |> map redshift_from_a |> p
```

reduce_axis

`reduce_axis` — batch 54 builtin.

```
my $result = reduce_axis $x
# or in a pipeline:
@list |> map reduce_axis |> p
```

reduce_indexed

`reduce_indexed` — go/general functional utilities builtin.

```
my $result = reduce_indexed $x
# or in a pipeline:
@list |> map reduce_indexed |> p
```

reduce_right

`reduce_right` — additional missing stdlib functions builtin.

```
my $result = reduce_right $x
# or in a pipeline:
@list |> map reduce_right |> p
```

refaddr

`refaddr` — list / aggregate builtin.

```
my $result = refaddr $x
# or in a pipeline:
@list |> map refaddr |> p
```

refl_check

`refl_check` — batch 67 builtin.

```
my $result = refl_check $x
# or in a pipeline:
@list |> map refl_check |> p
```

reflect_point

`reflect_point($x, $y, $axis)` — reflects a point across an axis. Axis can be 'x', 'y', or 'origin'. Returns the reflected coordinates.

```
p reflect_point(3, 4, 'x')       # [3, -4]
p reflect_point(3, 4, 'y')       # [-3, 4]
p reflect_point(3, 4, 'origin')  # [-3, -4]
```

reflpt

`reflpt` — geometry (extended) builtin. Alias for `reflect_point`.

```
my $result = reflpt $x
# or in a pipeline:
@list |> map reflpt |> p
```

refresh_stashes

`refresh_stashes` — symbol table builtin.

```
my $result = refresh_stashes $x
# or in a pipeline:
@list |> map refresh_stashes |> p
```

reftype

`reftype` — list / aggregate builtin.

```
my $result = reftype $x
# or in a pipeline:
@list |> map reftype |> p
```

regex_compile_thompson

`regex_compile_thompson` — batch 68 builtin.

```
my $result = regex_compile_thompson $x
# or in a pipeline:
@list |> map regex_compile_thompson |> p
```

regex_escape_simple

`regex_escape_simple` — batch 10 builtin.

```
my $result = regex_escape_simple $x
# or in a pipeline:
@list |> map regex_escape_simple |> p
```

regex_extract

`regex_extract` — file stat / path builtin.

```
my $result = regex_extract $x
# or in a pipeline:
@list |> map regex_extract |> p
```

regex_match_dfa

`regex_match_dfa` — batch 68 builtin.

```
my $result = regex_match_dfa $x
# or in a pipeline:
@list |> map regex_match_dfa |> p
```

regex_replace_str

`regex_replace_str` — file stat / path builtin.

```
my $result = regex_replace_str $x
# or in a pipeline:
@list |> map regex_replace_str |> p
```

regex_split_str

`regex_split_str` — file stat / path builtin.

```
my $result = regex_split_str $x
# or in a pipeline:
@list |> map regex_split_str |> p
```

regex_to_nfa_thompson

`regex_to_nfa_thompson` — batch 68 builtin.

```
my $result = regex_to_nfa_thompson $input
```

regexp_matches

`regexp_matches` — batch 70 builtin.

```
my $result = regexp_matches $x
# or in a pipeline:
@list |> map regexp_matches |> p
```

regexp_replace

regexp_replace — batch 70 builtin.

```
my $result = regexp_replace $x
# or in a pipeline:
@list |> map regexp_replace |> p
```

regexp_split

regexp_split — batch 70 builtin.

```
my $result = regexp_split $x
# or in a pipeline:
@list |> map regexp_split |> p
```

regr_intercept

regr_intercept — batch 70 builtin.

```
my $result = regr_intercept $x
# or in a pipeline:
@list |> map regr_intercept |> p
```

regr_r2

regr_r2 — batch 70 builtin.

```
my $result = regr_r2 $x
# or in a pipeline:
@list |> map regr_r2 |> p
```

regr_slope

regr_slope — batch 70 builtin.

```
my $result = regr_slope $x
# or in a pipeline:
@list |> map regr_slope |> p
```

regula_falsi

regula_falsi — batch 19 builtin.

```
my $result = regula_falsi $x
# or in a pipeline:
@list |> map regula_falsi |> p
```

regular_ngon_area

regular_ngon_area — batch 28 builtin.

```
my $result = regular_ngon_area $x
# or in a pipeline:
@list |> map regular_ngon_area |> p
```

regular_ngon_circumradius

regular_ngon_circumradius — batch 28 builtin.

```
my $result = regular_ngon_circumradius $x
# or in a pipeline:
@list |> map regular_ngon_circumradius |> p
```

regular_ngon_inradius

`regular_ngon_inradius` — batch 28 builtin.

```
my $result = regular_ngon_inradius $x
# or in a pipeline:
@list |> map regular_ngon_inradius |> p
```

regulator_naive

`regulator_naive` — batch 35 builtin.

```
my $result = regulator_naive $x
# or in a pipeline:
@list |> map regulator_naive |> p
```

reject_if

`reject_if` — functional combinators builtin.

```
my $result = reject_if $x
# or in a pipeline:
@list |> map reject_if |> p
```

rel_energy_pm

`rel_energy_pm` — batch 23 builtin.

```
my $result = rel_energy_pm $x
# or in a pipeline:
@list |> map rel_energy_pm |> p
```

rel_ke

`rel_ke` — batch 23 builtin.

```
my $result = rel_ke $x
# or in a pipeline:
@list |> map rel_ke |> p
```

rel_momentum

`rel_momentum` — batch 23 builtin.

```
my $result = rel_momentum $x
# or in a pipeline:
@list |> map rel_momentum |> p
```

rel_total_energy

`rel_total_energy` — batch 23 builtin.

```
my $result = rel_total_energy $x
# or in a pipeline:
@list |> map rel_total_energy |> p
```

rel_velocity_add

`rel_velocity_add` — batch 23 builtin.

```
my $result = rel_velocity_add $x
# or in a pipeline:
@list |> map rel_velocity_add |> p
```

relative_entropy_coherence

`relative_entropy_coherence` — batch 32 builtin.

```
my $result = relative_entropy_coherence $x
# or in a pipeline:
@list |> map relative_entropy_coherence |> p
```

relative_entropy_kl

`relative_entropy_kl` — batch 40 builtin.

```
my $result = relative_entropy_kl $x
# or in a pipeline:
@list |> map relative_entropy_kl |> p
```

relative_humidity_step

`relative_humidity_step` — batch 42 builtin.

```
my $result = relative_humidity_step $x
# or in a pipeline:
@list |> map relative_humidity_step |> p
```

relative_luminance

`relative_luminance` — batch 9 builtin.

```
my $result = relative_luminance $x
# or in a pipeline:
@list |> map relative_luminance |> p
```

relative_vorticity_step

`relative_vorticity_step` — batch 42 builtin.

```
my $result = relative_vorticity_step $x
# or in a pipeline:
@list |> map relative_vorticity_step |> p
```

relativistic_doppler

`relativistic_doppler` — batch 23 builtin.

```
my $result = relativistic_doppler $x
# or in a pipeline:
@list |> map relativistic_doppler |> p
```

relativistic_energy

`relativistic_energy` — physics formulas builtin.

```
my $result = relativistic_energy $x
# or in a pipeline:
@list |> map relativistic_energy |> p
```

relativistic_kinetic

`relativistic_kinetic` — batch 12 builtin.

```
my $result = relativistic_kinetic $x
# or in a pipeline:
@list |> map relativistic_kinetic |> p
```

relativistic_mass

`relativistic_mass` — physics formulas builtin.

```
my $result = relativistic_mass $x
# or in a pipeline:
@list |> map relativistic_mass |> p
```

relenergy

`relenergy` — physics formulas builtin. Alias for `relativistic_energy`.

```
my $result = relenergy $x
# or in a pipeline:
@list |> map relenergy |> p
```

reliability_diagram

`reliability_diagram` — batch 57 builtin.

```
my $result = reliability_diagram $x
# or in a pipeline:
@list |> map reliability_diagram |> p
```

relmass

`relmass` — physics formulas builtin. Alias for `relativistic_mass`.

```
my $result = relmass $x
# or in a pipeline:
@list |> map relmass |> p
```

relu

`relu` — math functions builtin.

```
my $result = relu $x
# or in a pipeline:
@list |> map relu |> p
```

relu_grad

`relu_grad` — batch 27 builtin.

```
my $result = relu_grad $x
# or in a pipeline:
@list |> map relu_grad |> p
```

remc

`remc` — algebraic match builtin. Alias for `remove_clj`.

```
my $result = remc $x
# or in a pipeline:
@list |> map remc |> p
```

remez_design

`remez_design` — batch 74 builtin.

```
my $result = remez_design $x
# or in a pipeline:
@list |> map remez_design |> p
```

remove_all_str

`remove_all_str` — extended stdlib builtin.

```
my $result = remove_all_str $x
# or in a pipeline:
@list |> map remove_all_str |> p
```

remove_at

`remove_at` — list helpers (batch 4) builtin.

```
my $result = remove_at $x
# or in a pipeline:
@list |> map remove_at |> p
```

remove_clj

`remove_clj` — algebraic match builtin.

```
my $result = remove_clj $x
# or in a pipeline:
@list |> map remove_clj |> p
```

remove_consonants

`remove_consonants` — string processing (uncategorized batch) builtin.

```
my $result = remove_consonants $x
# or in a pipeline:
@list |> map remove_consonants |> p
```

remove_elem

`remove_elem` — list helpers (batch 4) builtin.

```
my $result = remove_elem $x
# or in a pipeline:
@list |> map remove_elem |> p
```

remove_first_elem

`remove_first_elem` — list helpers (batch 4) builtin.

```
my $result = remove_first_elem $x
# or in a pipeline:
@list |> map remove_first_elem |> p
```

remove_keys

`remove_keys` — hash ops (batch 2) builtin.

```
my $result = remove_keys $x
# or in a pipeline:
@list |> map remove_keys |> p
```

remove_str

`remove_str` — string helpers (batch 4) builtin.

```
my $result = remove_str $x
# or in a pipeline:
@list |> map remove_str |> p
```

remove_vowels

`remove_vowels` — string processing (uncategorized batch) builtin.

```
my $result = remove_vowels $x
# or in a pipeline:
@list |> map remove_vowels |> p
```

remove_whitespace

`remove_whitespace` — extended stdlib builtin.

```
my $result = remove_whitespace $x
# or in a pipeline:
@list |> map remove_whitespace |> p
```

renamenx

`renamenx` — batch 71 builtin.

```
my $result = renamenx $x
# or in a pipeline:
@list |> map renamenx |> p
```

renyi_divergence_step

`renyi_divergence_step` — batch 40 builtin.

```
my $result = renyi_divergence_step $x
# or in a pipeline:
@list |> map renyi_divergence_step |> p
```

rep

`rep` — r base builtin. Alias for `rep_fn`.

```
my $result = rep $x
# or in a pipeline:
@list |> map rep |> p
```

rep_fn

`rep_fn` — repeat a value N times. Like R's rep().

```
my @r = @{rep_fn(0, 10)}   # ten zeros
my @s = @{rep_fn("x", 5)}  # five "x"s
```

repa

`repa` — javascript array/object methods builtin. Alias for `replace_at`.

```
my $result = repa $x
# or in a pipeline:
@list |> map repa |> p
```

repdigit

`repdigit` — batch 11 builtin.

```
my $result = repdigit $x
# or in a pipeline:
@list |> map repdigit |> p
```

repeat

`repeat` — trivial string ops builtin.

```
my $result = repeat $x
# or in a pipeline:
@list |> map repeat |> p
```

repeat_elem

`repeat_elem` — list helpers (batch 4) builtin.

```
my $result = repeat_elem $x
# or in a pipeline:
@list |> map repeat_elem |> p
```

repeat_list

`repeat_list` — collection (batch 2) builtin.

```
my $result = repeat_list $x
# or in a pipeline:
@list |> map repeat_list |> p
```

repeat_string

`repeat_string` — string helpers (batch 4) builtin.

```
my $result = repeat_string $x
# or in a pipeline:
@list |> map repeat_string |> p
```

repeated_game_avg_payoff

`repeated_game_avg_payoff` — batch 43 builtin.

```
my $result = repeated_game_avg_payoff $x
# or in a pipeline:
@list |> map repeated_game_avg_payoff |> p
```

repeatedly

`repeatedly` — algebraic match builtin.

```
my $result = repeatedly $x
# or in a pipeline:
@list |> map repeatedly |> p
```

replace_all_str

`replace_all_str` — string (batch 2) builtin.

```
my $result = replace_all_str $x
# or in a pipeline:
@list |> map replace_all_str |> p
```

replace_at

`replace_at` — javascript array/object methods builtin.

```
my $result = replace_at $x
# or in a pipeline:
@list |> map replace_at |> p
```

replace_at_index

`replace_at_index` — batch 9 builtin.

```
my $result = replace_at_index $x
# or in a pipeline:
@list |> map replace_at_index |> p
```

replace_first

`replace_first` — string (batch 2) builtin.

```
my $result = replace_first $x
# or in a pipeline:
@list |> map replace_first |> p
```

replace_n_times

`replace_n_times` — extended stdlib builtin.

```
my $result = replace_n_times $x
# or in a pipeline:
@list |> map replace_n_times |> p
```

replace_regex

`replace_regex` — extended stdlib builtin.

```
my $result = replace_regex $x
# or in a pipeline:
@list |> map replace_regex |> p
```

replicate_val

`replicate_val` — haskell list functions builtin.

```
my $result = replicate_val $x
# or in a pipeline:
@list |> map replicate_val |> p
```

replicator_dynamics_step

`replicator_dynamics_step` — batch 43 builtin.

```
my $result = replicator_dynamics_step $x
# or in a pipeline:
@list |> map replicator_dynamics_step |> p
```

repln

`repln` — extended stdlib builtin. Alias for `replace_n_times`.

```
my $result = repln $x
# or in a pipeline:
@list |> map repln |> p
```

replre

`replre` — extended stdlib builtin. Alias for `replace_regex`.

```
my $result = replre $x
# or in a pipeline:
@list |> map replre |> p
```

representation_dim_step

`representation_dim_step` — batch 37 builtin.

```
my $result = representation_dim_step $x
# or in a pipeline:
@list |> map representation_dim_step |> p
```

repunit

`repunit` — batch 11 builtin.

```
my $result = repunit $x
# or in a pipeline:
@list |> map repunit |> p
```

repv

`repv` — haskell list functions builtin. Alias for `replicate_val`.

```
my $result = repv $x
# or in a pipeline:
@list |> map repv |> p
```

resample

`resample(\@signal, $factor)` — resamples signal by a rational factor. Factor > 1 upsamples, factor < 1 downsamples. Uses linear interpolation.

```
my $up = resample(\@signal, 2)     # double sample rate
my $down = resample(\@signal, 0.5) # halve sample rate
```

resample_linear

`resample_linear` — batch 25 builtin.

```
my $result = resample_linear $x
# or in a pipeline:
@list |> map resample_linear |> p
```

resample_nearest

`resample_nearest` — batch 25 builtin.

```
my $result = resample_nearest $x
# or in a pipeline:
@list |> map resample_nearest |> p
```

resample_poly_step

`resample_poly_step` — batch 74 builtin.

```
my $result = resample_poly_step $x
# or in a pipeline:
@list |> map resample_poly_step |> p
```

reserve_prospective

`reserve_prospective` — batch 60 builtin.

```
my $result = reserve_prospective $x
# or in a pipeline:
@list |> map reserve_prospective |> p
```

reserve_retrospective

`reserve_retrospective` — batch 60 builtin.

```
my $result = reserve_retrospective $x
# or in a pipeline:
@list |> map reserve_retrospective |> p
```

reservoir_sample

`reservoir_sample` — matrix operations (uncategorized batch) builtin.

```
my $result = reservoir_sample $x
# or in a pipeline:
@list |> map reservoir_sample |> p
```

reset

`reset` — process / system builtin.

```
my $result = reset $x
# or in a pipeline:
@list |> map reset |> p
```

resfreq

`resfreq` — physics formulas builtin. Alias for `resonant_frequency`.

```
my $result = resfreq $x
# or in a pipeline:
@list |> map resfreq |> p
```

reshape_dim

`reshape_dim` — batch 54 builtin.

```
my $result = reshape_dim $x
# or in a pipeline:
@list |> map reshape_dim |> p
```

residuals_compute

`residuals_compute` — batch 9 builtin.

```
my $result = residuals_compute $x
# or in a pipeline:
@list |> map residuals_compute |> p
```

resistance_parallel

`resistance_parallel` — batch 12 builtin.

```
my $result = resistance_parallel $x
# or in a pipeline:
@list |> map resistance_parallel |> p
```

resistance_series

`resistance_series` — batch 12 builtin.

```
my $result = resistance_series $x
# or in a pipeline:
@list |> map resistance_series |> p
```

resolution_step

`resolution_step` — batch 67 builtin.

```
my $result = resolution_step $x
# or in a pipeline:
@list |> map resolution_step |> p
```

resonance_count

`resonance_count` — batch 63 builtin.

```
my $result = resonance_count $x
# or in a pipeline:
@list |> map resonance_count |> p
```

resonant_frequency

`resonant_frequency` — physics formulas builtin.

```
my $result = resonant_frequency $x
# or in a pipeline:
@list |> map resonant_frequency |> p
```

rest

`rest` — algebraic match builtin.

```
my $result = rest $x
# or in a pipeline:
@list |> map rest |> p
```

rest_energy

`rest_energy` — physics formulas builtin.

```
my $result = rest_energy $x
# or in a pipeline:
@list |> map rest_energy |> p
```

restenergy

`restenergy` — physics formulas builtin. Alias for `rest_energy`.

```
my $result = restenergy $x
# or in a pipeline:
@list |> map restenergy |> p
```

rev

`rev` — short aliases builtin.

```
my $result = rev $x
# or in a pipeline:
@list |> map rev |> p
```

rev_str

`rev_str` — trivial string ops builtin. Alias for `reverse_str`.

```
my $result = rev_str $x
# or in a pipeline:
@list |> map rev_str |> p
```

rev_vec

`rev_vec` — r base builtin.

```
my $result = rev_vec $x
# or in a pipeline:
@list |> map rev_vec |> p
```

revenue_equivalence_check

`revenue_equivalence_check` — batch 43 builtin.

```
my $result = revenue_equivalence_check $x
# or in a pipeline:
@list |> map revenue_equivalence_check |> p
```

reverse_delete_step

`reverse_delete_step` — batch 75 builtin.

```
my $result = reverse_delete_step $x
# or in a pipeline:
@list |> map reverse_delete_step |> p
```

reverse_each

`reverse_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = reverse_each $x
# or in a pipeline:
@list |> map reverse_each |> p
```

reverse_each_word

`reverse_each_word` — string processing (uncategorized batch) builtin.

```
my $result = reverse_each_word $x
# or in a pipeline:
@list |> map reverse_each_word |> p
```

reverse_list

`reverse_list` — file stat / path builtin.

```
my $result = reverse_list $x
# or in a pipeline:
@list |> map reverse_list |> p
```

reverse_str

`reverse_str` — trivial string ops builtin.

```
my $result = reverse_str $x
# or in a pipeline:
@list |> map reverse_str |> p
```

reverse_transcribe

`reverse_transcribe` — batch 33 builtin.

```
my $result = reverse_transcribe $x
# or in a pipeline:
@list |> map reverse_transcribe |> p
```

reverse_words

`reverse_words` — string (batch 2) builtin.

```
my $result = reverse_words $x
# or in a pipeline:
@list |> map reverse_words |> p
```

revwords

`revwords` — string processing (uncategorized batch) builtin. Alias for `reverse_each_word`.

```
my $result = revwords $x
# or in a pipeline:
@list |> map revwords |> p
```

rexp

`rexp` — generate n random exponential variates. Args: n [, rate]. Like R's rexp().

```
my @x = @{rexp(100, 0.5)}       # rate=0.5 (mean=2)
```

reynolds_full_number

`reynolds_full_number` — batch 42 builtin.

```
my $result = reynolds_full_number $x
# or in a pipeline:
@list |> map reynolds_full_number |> p
```

rf

`rf` — generate n random F variates. Args: n, d1, d2. Like R's rf().

```
my @x = @{rf(100, 5, 10)}       # F(5,10)
```

rfact

`rfact` — math formulas builtin. Alias for `rising_factorial`.

```
my $result = rfact $x
# or in a pipeline:
@list |> map rfact |> p
```

rfc3339

`rfc3339` — batch 10 builtin. Alias for `rfc3339_format`.

```
my $result = rfc3339 $x
# or in a pipeline:
@list |> map rfc3339 |> p
```

rfc3339_format

`rfc3339_format` — batch 10 builtin.

```
my $result = rfc3339_format $x
# or in a pipeline:
@list |> map rfc3339_format |> p
```

rfc3339_parse

`rfc3339_parse` — batch 10 builtin.

```
my $result = rfc3339_parse $x
# or in a pipeline:
@list |> map rfc3339_parse |> p
```

rg_radius_of_gyration

`rg_radius_of_gyration` — batch 63 builtin.

```
my $result = rg_radius_of_gyration $x
# or in a pipeline:
@list |> map rg_radius_of_gyration |> p
```

rgamma

`rgamma` — generate n random gamma variates. Args: n, shape [, scale]. Like R's rgamma().

```
my @x = @{rgamma(100, 2, 1)}    # Gamma(2,1)
```

rgb

`rgb` — color / ansi builtin.

```
my $result = rgb $x
# or in a pipeline:
@list |> map rgb |> p
```

rgb2hsl

`rgb2hsl` — color operations builtin. Alias for `rgb_to_hsl`.

```
my $result = rgb2hsl $x
# or in a pipeline:
@list |> map rgb2hsl |> p
```

rgb2hsv

`rgb2hsv` — color operations builtin. Alias for `rgb_to_hsv`.

```
my $result = rgb2hsv $x
# or in a pipeline:
@list |> map rgb2hsv |> p
```

rgb_to_cmyk

`rgb_to_cmyk` — batch 82 builtin.

```
my $result = rgb_to_cmyk $input
```

rgb_to_hex

`rgb_to_hex` — color / ansi builtin.

```
my $result = rgb_to_hex $input
```

rgb_to_hsl

`rgb_to_hsl` — color operations builtin.

```
my $result = rgb_to_hsl $input
```

rgb_to_hsv

`rgb_to_hsv` — color operations builtin.

```
my $result = rgb_to_hsv $input
```

rgb_to_oklab

`rgb_to_oklab` — batch 82 builtin.

```
my $result = rgb_to_oklab $input
```

rgb_to_xyz

`rgb_to_xyz` — batch 82 builtin.

```
my $result = rgb_to_xyz $input
```

rgb_to_yiq

`rgb_to_yiq` — batch 30 builtin.

```
my $result = rgb_to_yiq $input
```

rgb_to_yuv

`rgb_to_yuv` — batch 82 builtin.

```
my $result = rgb_to_yuv $input
```

rgb_to_yuv601

`rgb_to_yuv601` — batch 30 builtin.

```
my $result = rgb_to_yuv601 $input
```

rgeom

`rgeom` — generate n random geometric variates. Args: n, prob.

```
my @x = @{rgeom(100, 0.3)}      # trials until first success
```

rho

`spearman` (aliases `spearman_corr`, `rho`) computes Spearman's rank correlation coefficient between two samples.

```
p rho([1,2,3,4,5], [5,6,7,8,7])  # ~0.82
```

rhumb_line_bearing

`rhumb_line_bearing` — batch 58 builtin.

```
my $result = rhumb_line_bearing $x
# or in a pipeline:
@list |> map rhumb_line_bearing |> p
```

ricci_identity_step

`ricci_identity_step` — batch 39 builtin.

```
my $result = ricci_identity_step $x
# or in a pipeline:
@list |> map ricci_identity_step |> p
```

ricci_tensor_step_zero

`ricci_tensor_step_zero` — batch 39 builtin.

```
my $result = ricci_tensor_step_zero $x
# or in a pipeline:
@list |> map ricci_tensor_step_zero |> p
```

richardson_number_step

`richardson_number_step` — batch 42 builtin.

```
my $result = richardson_number_step $x
# or in a pipeline:
@list |> map richardson_number_step |> p
```

richter_local_ml

`richter_local_ml` — batch 65 builtin.

```
my $result = richter_local_ml $x
# or in a pipeline:
@list |> map richter_local_ml |> p
```

ricker_wavelet

`ricker_wavelet` — batch 74 builtin.

```
my $result = ricker_wavelet $x
# or in a pipeline:
@list |> map ricker_wavelet |> p
```

ridders

`ridders` — batch 19 builtin. Alias for `ridders_root`.

```
my $result = ridders $x
# or in a pipeline:
@list |> map ridders |> p
```

ridders_root

`ridders_root` — batch 19 builtin.

```
my $result = ridders_root $x
# or in a pipeline:
@list |> map ridders_root |> p
```

ridx

`ridx` — python/ruby stdlib builtin. Alias for `rindex_fn`.

```
my $result = ridx $x
# or in a pipeline:
@list |> map ridx |> p
```

riemann_curvature_normal_form

`riemann_curvature_normal_form` — batch 39 builtin.

```
my $result = riemann_curvature_normal_form $x
# or in a pipeline:
@list |> map riemann_curvature_normal_form |> p
```

riemann_tensor_step_zero

`riemann_tensor_step_zero` — batch 39 builtin.

```
my $result = riemann_tensor_step_zero $x
# or in a pipeline:
@list |> map riemann_tensor_step_zero |> p
```

riemann_xi

`riemann_xi` — batch 29 builtin.

```
my $result = riemann_xi $x
# or in a pipeline:
@list |> map riemann_xi |> p
```

riffle

`riffle` — collection helpers (trivial) builtin.

```
my $result = riffle $x
# or in a pipeline:
@list |> map riffle |> p
```

right_str

`right_str` — string (batch 2) builtin.

```
my $result = right_str $x
# or in a pipeline:
@list |> map right_str |> p
```

rindex_fn

`rindex_fn` — python/ruby stdlib builtin.

```
my $result = rindex_fn $x
# or in a pipeline:
@list |> map rindex_fn |> p
```

rising_factorial

`rising_factorial` — math formulas builtin.

```
my $result = rising_factorial $x
# or in a pipeline:
@list |> map rising_factorial |> p
```

risk_ratio_2x2

`risk_ratio_2x2` — batch 61 builtin.

```
my $result = risk_ratio_2x2 $x
# or in a pipeline:
@list |> map risk_ratio_2x2 |> p
```

rjt

`rjt` — extended stdlib builtin. Alias for `rjust_text`.

```
my $result = rjt $x
# or in a pipeline:
@list |> map rjt |> p
```

rjust

`rjust` — string helpers (batch 4) builtin.

```
my $result = rjust $x
# or in a pipeline:
@list |> map rjust |> p
```

rjust_text

`rjust_text` — extended stdlib builtin.

```
my $result = rjust_text $x
# or in a pipeline:
@list |> map rjust_text |> p
```

rk4

`rk4` (aliases `runge_kutta`, `rk4_ode`) solves an ODE dy/dt = f(t,y) using 4th-order Runge-Kutta. Returns [[t,y], ...].

```
# dy/dt = -y, y(0) = 1 □ y = e^(-t)
my $sol = rk4(fn { -$_[1] }, 0, 1, 0.1, 100)
```

rk45_cash_karp

`rk45_cash_karp` — batch 19 builtin.

```
my $result = rk45_cash_karp $x
# or in a pipeline:
@list |> map rk45_cash_karp |> p
```

rk4_single

`rk4_single` — batch 34 builtin.

```
my $result = rk4_single $x
# or in a pipeline:
@list |> map rk4_single |> p
```

rk_combine

`rk_combine` — batch 34 builtin.

```
my $result = rk_combine $x
# or in a pipeline:
@list |> map rk_combine |> p
```

rkck

`rkck` — batch 19 builtin. Alias for `rk45_cash_karp`.

```
my $result = rkck $x
# or in a pipeline:
@list |> map rkck |> p
```

rkf45_error

`rkf45_error` — batch 34 builtin.

```
my $result = rkf45_error $x
# or in a pipeline:
@list |> map rkf45_error |> p
```

rl_tau

`rl_tau` — batch 23 builtin.

```
my $result = rl_tau $x
# or in a pipeline:
@list |> map rl_tau |> p
```

rl_time_constant

`rl_time_constant` — physics formulas builtin.

```
my $result = rl_time_constant $x
# or in a pipeline:
@list |> map rl_time_constant |> p
```

rld

`rld` — collection more builtin. Alias for `run_length_decode`.

```
my $result = rld $x
# or in a pipeline:
@list |> map rld |> p
```

rld_str

`rld_str` — matrix operations (uncategorized batch) builtin. Alias for `run_length_decode_str`.

```
my $result = rld_str $x
# or in a pipeline:
@list |> map rld_str |> p
```

rle

`rle` — collection more builtin. Alias for `run_length_encode`.

```
my $result = rle $x
# or in a pipeline:
@list |> map rle |> p
```

rle_str

`rle_str` — matrix operations (uncategorized batch) builtin. Alias for `run_length_encode_str`.

```
my $result = rle_str $x
# or in a pipeline:
@list |> map rle_str |> p
```

rlnorm

`rlnorm` — generate n random log-normal variates. Args: n [, mu, sigma].

```
my @x = @{rlnorm(100, 0, 1)}    # LogN(0,1)
```

rltau

`rltau` — physics formulas builtin. Alias for `rl_time_constant`.

```
my $result = rltau $x
# or in a pipeline:
@list |> map rltau |> p
```

rmall

`rmall` — extended stdlib builtin. Alias for `remove_all_str`.

```
my $result = rmall $x
# or in a pipeline:
@list |> map rmall |> p
```

rmcons

`rmcons` — string processing (uncategorized batch) builtin. Alias for `remove_consonants`.

```
my $result = rmcons $x
# or in a pipeline:
@list |> map rmcons |> p
```

rms

`rms` — extended stdlib builtin.

```
my $result = rms $x
# or in a pipeline:
@list |> map rms |> p
```

rmse

`rmse` — math functions builtin.

```
my $result = rmse $x
# or in a pipeline:
@list |> map rmse |> p
```

rmvowel

`rmvowel` — string processing (uncategorized batch) builtin. Alias for `remove_vowels`.

```
my $result = rmvowel $x
# or in a pipeline:
@list |> map rmvowel |> p
```

rmws

`rmws` — extended stdlib builtin. Alias for `remove_whitespace`.

```
my $result = rmws $x
# or in a pipeline:
@list |> map rmws |> p
```

rngcmp

`rngcmp` — matrix operations (uncategorized batch) builtin. Alias for `range_compress`.

```
my $result = rngcmp $x
# or in a pipeline:
@list |> map rngcmp |> p
```

rngexp

`rngexp` — matrix operations (uncategorized batch) builtin. Alias for `range_expand`.

```
my $result = rngexp $x
# or in a pipeline:
@list |> map rngexp |> p
```

rnk

`rnk` — extended stdlib builtin. Alias for `rank`.

```
my $result = rnk $x
# or in a pipeline:
@list |> map rnk |> p
```

rnorm

rnorm — generate n random normal variates. Args: n [, mu, sigma]. Like R's rnorm().

```
my @x = @{rnorm(1000)}          # 1000 standard normal
my @y = @{rnorm(100, 50, 10)}   # mean=50, sd=10
```

roberts_kernel_value

roberts_kernel_value — batch 40 builtin.

```
my $result = roberts_kernel_value $x
# or in a pipeline:
@list |> map roberts_kernel_value |> p
```

robinson_project

robinson_project — batches 14-16 builtin.

```
my $result = robinson_project $x
# or in a pipeline:
@list |> map robinson_project |> p
```

robinson_schensted_pair

robinson_schensted_pair — batch 37 builtin.

```
my $result = robinson_schensted_pair $x
# or in a pipeline:
@list |> map robinson_schensted_pair |> p
```

robust_scale

robust_scale — batch 27 builtin.

```
my $result = robust_scale $x
# or in a pipeline:
@list |> map robust_scale |> p
```

robust_se_huber_white

robust_se_huber_white — batch 36 builtin.

```
my $result = robust_se_huber_white $x
# or in a pipeline:
@list |> map robust_se_huber_white |> p
```

roche_limit_fluid

roche_limit_fluid — batch 31 builtin.

```
my $result = roche_limit_fluid $x
# or in a pipeline:
@list |> map roche_limit_fluid |> p
```

roche_limit_rigid

roche_limit_rigid — batch 31 builtin.

```
my $result = roche_limit_rigid $x
# or in a pipeline:
@list |> map roche_limit_rigid |> p
```

roi

`roi` — finance builtin.

```
my $result = roi $x
# or in a pipeline:
@list |> map roi |> p
```

roll_n

`roll_n` — batch 54 builtin.

```
my $result = roll_n $x
# or in a pipeline:
@list |> map roll_n |> p
```

roman

`roman` — roman numerals builtin. Alias for `int_to_roman`.

```
my $result = roman $x
# or in a pipeline:
@list |> map roman |> p
```

roman_add

`roman_add` — roman numerals builtin.

```
my $result = roman_add $x
# or in a pipeline:
@list |> map roman_add |> p
```

roman_decode

`roman_decode` — batch 30 builtin.

```
my $result = roman_decode $x
# or in a pipeline:
@list |> map roman_decode |> p
```

roman_encode

`roman_encode` — batch 30 builtin.

```
my $result = roman_encode $x
# or in a pipeline:
@list |> map roman_encode |> p
```

roman_numeral_list

`roman_numeral_list` — roman numerals builtin.

```
my $result = roman_numeral_list $x
# or in a pipeline:
@list |> map roman_numeral_list |> p
```

roman_to_int

`roman_to_int` — roman numerals builtin.

```
my $result = roman_to_int $input
```

romanlist

`romanlist` — roman numerals builtin. Alias for `roman_numeral_list`.

```
my $result = romanlist $x
# or in a pipeline:
@list |> map romanlist |> p
```

romberg_quad

`romberg_quad` — batch 72 builtin.

```
my $result = romberg_quad $x
# or in a pipeline:
@list |> map romberg_quad |> p
```

roommate_match_step

`roommate_match_step` — batch 43 builtin.

```
my $result = roommate_match_step $x
# or in a pipeline:
@list |> map roommate_match_step |> p
```

root_brentq

`root_brentq` — batch 72 builtin.

```
my $result = root_brentq $x
# or in a pipeline:
@list |> map root_brentq |> p
```

root_newton

`root_newton` — batch 72 builtin.

```
my $result = root_newton $x
# or in a pipeline:
@list |> map root_newton |> p
```

root_secant

`root_secant` — batch 72 builtin.

```
my $result = root_secant $x
# or in a pipeline:
@list |> map root_secant |> p
```

root_system_count

`root_system_count` — batch 37 builtin.

```
my $result = root_system_count $x
# or in a pipeline:
@list |> map root_system_count |> p
```

ros2_step

`ros2_step` — batch 34 builtin.

```
my $result = ros2_step $x
# or in a pipeline:
@list |> map ros2_step |> p
```

rot13

`rot13` — string (batch 2) builtin.

```
my $result = rot13 $x
# or in a pipeline:
@list |> map rot13 |> p
```

rot47

`rot47` — string (batch 2) builtin.

```
my $result = rot47 $x
# or in a pipeline:
@list |> map rot47 |> p
```

rotate

`rotate` — collection helpers (trivial) builtin.

```
my $result = rotate $x
# or in a pipeline:
@list |> map rotate |> p
```

rotate_axis

`rotate_axis` — batch 54 builtin.

```
my $result = rotate_axis $x
# or in a pipeline:
@list |> map rotate_axis |> p
```

rotate_point

`rotate_point` — math functions builtin.

```
my $result = rotate_point $x
# or in a pipeline:
@list |> map rotate_point |> p
```

roulette_wheel_select_index

`roulette_wheel_select_index` — batch 41 builtin.

```
my $result = roulette_wheel_select_index $x
# or in a pipeline:
@list |> map roulette_wheel_select_index |> p
```

round

`round` — trivial numeric / predicate builtins builtin.

```
my $result = round $x
# or in a pipeline:
@list |> map round |> p
```

round_each

`round_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = round_each $x
# or in a pipeline:
@list |> map round_each |> p
```

round_to

round_to — math functions builtin.

```
my $result = round_to $x
# or in a pipeline:
@list |> map round_to |> p
```

rowMeans

row_means (alias rowMeans) — mean of each row. Like R's rowMeans().

```
p rowMeans([[2,4],[6,8]])  # [3, 7]
```

rowSums

row_sums (alias rowSums) — sum of each row in a matrix. Like R's rowSums().

```
p row_sums([[1,2,3],[4,5,6]])  # [6, 15]
```

rpad

rpad — trivial string ops builtin. Alias for pad_right.

```
my $result = rpad $x
# or in a pipeline:
@list |> map rpad |> p
```

rpn

rpn — algorithms / puzzles builtin. Alias for eval_rpn.

```
my $result = rpn $x
# or in a pipeline:
@list |> map rpn |> p
```

rpois

rpois — generate n random Poisson variates. Args: n, lambda. Like R's rpois().

```
my @x = @{rpois(100, 5)}        # Poisson with mean 5
```

rptd

rptd — algebraic match builtin. Alias for repeatedly.

```
my $result = rptd $x
# or in a pipeline:
@list |> map rptd |> p
```

rpush

rpush — batch 71 builtin.

```
my $result = rpush $x
# or in a pipeline:
@list |> map rpush |> p
```

rrt_extend

`rrt_extend` — batch 59 builtin.

```
my $result = rrt_extend $x
# or in a pipeline:
@list |> map rrt_extend |> p
```

rrt_star_rewire

`rrt_star_rewire` — batch 59 builtin.

```
my $result = rrt_star_rewire $x
# or in a pipeline:
@list |> map rrt_star_rewire |> p
```

rrule_next_occurrence

`rrule_next_occurrence` — batch 82 builtin.

```
my $result = rrule_next_occurrence $x
# or in a pipeline:
@list |> map rrule_next_occurrence |> p
```

rsa_d_from_e

`rsa_d_from_e` — batch 35 builtin.

```
my $result = rsa_d_from_e $x
# or in a pipeline:
@list |> map rsa_d_from_e |> p
```

rsa_decrypt_pkcs1

`rsa_decrypt_pkcs1` decrypts RSA-PKCS1v15 ciphertext.

```
my $plain = rsa_decrypt_pkcs1($priv, $cipher)
```

rsa_encrypt_pkcs1

`rsa_encrypt_pkcs1` encrypts with legacy RSA-PKCS1v15 padding. Only for compatibility with old systems — prefer OAEP.

```
my $cipher = rsa_encrypt_pkcs1($pub, "data")
```

rsa_encrypt_simple

`rsa_encrypt_simple` — batch 26 builtin.

```
my $result = rsa_encrypt_simple $x
# or in a pipeline:
@list |> map rsa_encrypt_simple |> p
```

rsamp

`rsamp` — matrix operations (uncategorized batch) builtin. Alias for `reservoir_sample`.

```
my $result = rsamp $x
# or in a pipeline:
@list |> map rsamp |> p
```

rsk_size

`rsk_size` — batch 37 builtin.

```
my $result = rsk_size $x
# or in a pipeline:
@list |> map rsk_size |> p
```

rst

`rst` — algebraic match builtin. Alias for `rest`.

```
my $result = rst $x
# or in a pipeline:
@list |> map rst |> p
```

rsun

`rsun` — physics constants builtin. Alias for `sun_radius`.

```
my $result = rsun $x
# or in a pipeline:
@list |> map rsun |> p
```

rt

`rt` — generate n random Student's t variates. Args: n, df. Like R's rt().

```
my @x = @{rt(100, 10)}          # t with 10 df
```

rt_effective

`rt_effective` — batch 22 builtin.

```
my $result = rt_effective $x
# or in a pipeline:
@list |> map rt_effective |> p
```

rt_serial_interval

`rt_serial_interval` — batch 61 builtin.

```
my $result = rt_serial_interval $x
# or in a pipeline:
@list |> map rt_serial_interval |> p
```

ruin_probability_lundberg

`ruin_probability_lundberg` — batch 60 builtin.

```
my $result = ruin_probability_lundberg $x
# or in a pipeline:
@list |> map ruin_probability_lundberg |> p
```

run_length_decode

`run_length_decode` — collection more builtin.

```
my $result = run_length_decode $x
# or in a pipeline:
@list |> map run_length_decode |> p
```

run_length_decode_str

run_length_decode_str — matrix operations (uncategorized batch) builtin.

```
my $result = run_length_decode_str $x
# or in a pipeline:
@list |> map run_length_decode_str |> p
```

run_length_encode

run_length_encode — collection more builtin.

```
my $result = run_length_encode $x
# or in a pipeline:
@list |> map run_length_encode |> p
```

run_length_encode_str

run_length_encode_str — matrix operations (uncategorized batch) builtin.

```
my $result = run_length_encode_str $x
# or in a pipeline:
@list |> map run_length_encode_str |> p
```

run_off_triangle_step

run_off_triangle_step — batch 60 builtin.

```
my $result = run_off_triangle_step $x
# or in a pipeline:
@list |> map run_off_triangle_step |> p
```

runif

runif — generate n random uniform variates. Args: n [, min, max]. Like R's runif().

```
my @x = @{runif(100, 0, 1)}     # 100 uniform [0,1]
```

running_max

running_max — list helpers (batch 4) builtin.

```
my $result = running_max $x
# or in a pipeline:
@list |> map running_max |> p
```

running_mean

running_mean — batch 11 builtin.

```
my $result = running_mean $x
# or in a pipeline:
@list |> map running_mean |> p
```

running_min

running_min — list helpers (batch 4) builtin.

```
my $result = running_min $x
# or in a pipeline:
@list |> map running_min |> p
```

running_reduce

`running_reduce` — additional missing stdlib functions builtin.

```
my $result = running_reduce $x
# or in a pipeline:
@list |> map running_reduce |> p
```

running_variance

`running_variance` — batch 11 builtin.

```
my $result = running_variance $x
# or in a pipeline:
@list |> map running_variance |> p
```

runred

`runred` — additional missing stdlib functions builtin. Alias for `running_reduce`.

```
my $result = runred $x
# or in a pipeline:
@list |> map runred |> p
```

rweibull

`rweibull` — generate n random Weibull variates. Args: n, shape [, scale].

```
my @x = @{rweibull(100, 2, 1)}  # Weibull(2,1)
```

rx_gate

`rx_gate` — batches 14-16 builtin.

```
my $result = rx_gate $x
# or in a pipeline:
@list |> map rx_gate |> p
```

rxn_q

`rxn_q` — batch 21 builtin. Alias for `reaction_quotient`.

```
my $result = rxn_q $x
# or in a pipeline:
@list |> map rxn_q |> p
```

ry_gate

`ry_gate` — batches 14-16 builtin.

```
my $result = ry_gate $x
# or in a pipeline:
@list |> map ry_gate |> p
```

rydberg

`rydberg` — physics constants builtin. Alias for `rydberg_constant`.

```
my $result = rydberg $x
# or in a pipeline:
@list |> map rydberg |> p
```

rydberg_constant

`rydberg_constant` — physics constants builtin.

```
my $result = rydberg_constant $x
# or in a pipeline:
@list |> map rydberg_constant |> p
```

rydberg_lambda

`rydberg_lambda` — batch 21 builtin.

```
my $result = rydberg_lambda $x
# or in a pipeline:
@list |> map rydberg_lambda |> p
```

ryu_takayanagi_step

`ryu_takayanagi_step` — batch 39 builtin.

```
my $result = ryu_takayanagi_step $x
# or in a pipeline:
@list |> map ryu_takayanagi_step |> p
```

rz_gate

`rz_gate` — batches 14-16 builtin.

```
my $result = rz_gate $x
# or in a pipeline:
@list |> map rz_gate |> p
```

s1

`s1` — crypto / encoding builtin. Alias for `sha1`.

```
my $result = s1 $x
# or in a pipeline:
@list |> map s1 |> p
```

s224

`s224` — crypto / encoding builtin. Alias for `sha224`.

```
my $result = s224 $x
# or in a pipeline:
@list |> map s224 |> p
```

s256

`s256` — crypto / encoding builtin. Alias for `sha256`.

```
my $result = s256 $x
# or in a pipeline:
@list |> map s256 |> p
```

s2_cell_at_lat_lng

`s2_cell_at_lat_lng` — batch 58 builtin.

```
my $result = s2_cell_at_lat_lng $x
# or in a pipeline:
@list |> map s2_cell_at_lat_lng |> p
```

s2_cell_id

`s2_cell_id` — batch 58 builtin.

```
my $result = s2_cell_id $x
# or in a pipeline:
@list |> map s2_cell_id |> p
```

s2_cell_neighbors

`s2_cell_neighbors` — batch 58 builtin.

```
my $result = s2_cell_neighbors $x
# or in a pipeline:
@list |> map s2_cell_neighbors |> p
```

s2c

`s2c` — string processing (uncategorized batch) builtin. Alias for `snake_to_camel`.

```
my $result = s2c $x
# or in a pipeline:
@list |> map s2c |> p
```

s384

`s384` — crypto / encoding builtin. Alias for `sha384`.

```
my $result = s384 $x
# or in a pipeline:
@list |> map s384 |> p
```

s512

`s512` — crypto / encoding builtin. Alias for `sha512`.

```
my $result = s512 $x
# or in a pipeline:
@list |> map s512 |> p
```

s_to_m

`s_to_m` — unit conversions builtin. Alias for `seconds_to_minutes`.

```
my $result = s_to_m $input
```

s_to_ms

`s_to_ms` — file stat / path builtin.

```
my $result = s_to_ms $input
```

sa_accept_prob

`sa_accept_prob` — batch 19 builtin.

```
my $result = sa_accept_prob $x
# or in a pipeline:
@list |> map sa_accept_prob |> p
```

sa_boltzmann_temp

sa_boltzmann_temp — batch 19 builtin.

```
my $result = sa_boltzmann_temp $x
# or in a pipeline:
@list |> map sa_boltzmann_temp |> p
```

sa_cauchy_temp

sa_cauchy_temp — batch 19 builtin.

```
my $result = sa_cauchy_temp $x
# or in a pipeline:
@list |> map sa_cauchy_temp |> p
```

sa_geometric_temp

sa_geometric_temp — batch 19 builtin.

```
my $result = sa_geometric_temp $x
# or in a pipeline:
@list |> map sa_geometric_temp |> p
```

sabr_implied_vol

sabr_implied_vol — batch 20 builtin.

```
my $result = sabr_implied_vol $x
# or in a pipeline:
@list |> map sabr_implied_vol |> p
```

sadd

sadd — batch 71 builtin.

```
my $result = sadd $x
# or in a pipeline:
@list |> map sadd |> p
```

saddle_point_check

saddle_point_check — batch 43 builtin.

```
my $result = saddle_point_check $x
# or in a pipeline:
@list |> map saddle_point_check |> p
```

safe_div

safe_div — functional combinators builtin.

```
my $result = safe_div $x
# or in a pipeline:
@list |> map safe_div |> p
```

safe_log

`safe_log` — functional combinators builtin.

```
my $result = safe_log $x
# or in a pipeline:
@list |> map safe_log |> p
```

safe_mod

`safe_mod` — functional combinators builtin.

```
my $result = safe_mod $x
# or in a pipeline:
@list |> map safe_mod |> p
```

safe_sqrt

`safe_sqrt` — functional combinators builtin.

```
my $result = safe_sqrt $x
# or in a pipeline:
@list |> map safe_sqrt |> p
```

salloc

`salloc` — stress / telemetry extensions builtin. Alias for `stress_alloc`.

```
my $result = salloc $x
# or in a pipeline:
@list |> map salloc |> p
```

salt_activity_coefficient

`salt_activity_coefficient` — batch 38 builtin.

```
my $result = salt_activity_coefficient $x
# or in a pipeline:
@list |> map salt_activity_coefficient |> p
```

sample_n

`sample_n` — collection (batch 2) builtin. Alias for `random_sample`.

```
my $result = sample_n $x
# or in a pipeline:
@list |> map sample_n |> p
```

sample_one

`sample_one` — list helpers (batch 4) builtin.

```
my $result = sample_one $x
# or in a pipeline:
@list |> map sample_one |> p
```

sample_stddev

`sample_stddev` — statistics (extended) builtin.

```
my $result = sample_stddev $x
# or in a pipeline:
@list |> map sample_stddev |> p
```

sample_variance

`sample_variance` — statistics (extended) builtin.

```
my $result = sample_variance $x
# or in a pipeline:
@list |> map sample_variance |> p
```

sampn

`sampn` — extended stdlib builtin.

```
my $result = sampn $x
# or in a pipeline:
@list |> map sampn |> p
```

sapply

`sapply` — apply a function to each element, return a vector. Like R's sapply().

```
my @sq = @{sapply([1,2,3,4], fn { $_[0] ** 2 })}  # [1,4,9,16]
```

sarimax_fit

`sarimax_fit` — batch 78 builtin.

```
my $result = sarimax_fit $x
# or in a pipeline:
@list |> map sarimax_fit |> p
```

saros_cycle

`saros_cycle` — batch 55 builtin.

```
my $result = saros_cycle $x
# or in a pipeline:
@list |> map saros_cycle |> p
```

sarsa_update

`sarsa_update` — batch 40 builtin.

```
my $result = sarsa_update $x
# or in a pipeline:
@list |> map sarsa_update |> p
```

sat01

`sat01` — trig / math (batch 2) builtin. Alias for `saturate`.

```
my $result = sat01 $x
# or in a pipeline:
@list |> map sat01 |> p
```

saturate

`saturate` — trig / math (batch 2) builtin.

```
my $result = saturate $x
# or in a pipeline:
@list |> map saturate |> p
```

saturation_equivalent_pt

`saturation_equivalent_pt` — batch 42 builtin.

```
my $result = saturation_equivalent_pt $x
# or in a pipeline:
@list |> map saturation_equivalent_pt |> p
```

saturation_vapor_pressure

`saturation_vapor_pressure` — batch 12 builtin.

```
my $result = saturation_vapor_pressure $x
# or in a pipeline:
@list |> map saturation_vapor_pressure |> p
```

save_pct

`save_pct` — batch 56 builtin.

```
my $result = save_pct $x
# or in a pipeline:
@list |> map save_pct |> p
```

savgol_coef

`savgol_coef` — batch 74 builtin.

```
my $result = savgol_coef $x
# or in a pipeline:
@list |> map savgol_coef |> p
```

sbranch

`sbranch` — stress / telemetry extensions builtin. Alias for `stress_branch`.

```
my $result = sbranch $x
# or in a pipeline:
@list |> map sbranch |> p
```

scalar_curvature_step

`scalar_curvature_step` — batch 39 builtin.

```
my $result = scalar_curvature_step $x
# or in a pipeline:
@list |> map scalar_curvature_step |> p
```

scale

`scale` — standardize a vector: (x - mean) / sd. Like R's scale().

```
my @z = @{scale([10,20,30])}  # [-1, 0, 1]
```

scale_each

`scale_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = scale_each $x
# or in a pipeline:
@list |> map scale_each |> p
```

scale_factor

`scale_factor` — batch 31 builtin.

```
my $result = scale_factor $x
# or in a pipeline:
@list |> map scale_factor |> p
```

scale_pitches_major

`scale_pitches_major` — batch 64 builtin.

```
my $result = scale_pitches_major $x
# or in a pipeline:
@list |> map scale_pitches_major |> p
```

scale_pitches_minor

`scale_pitches_minor` — batch 64 builtin.

```
my $result = scale_pitches_minor $x
# or in a pipeline:
@list |> map scale_pitches_minor |> p
```

scale_point

`scale_point($x, $y, $sx, $sy)` — scales a 2D point by factors sx and sy from the origin. Returns [$x*$sx, $y*$sy]. Use for geometric transformations.

```
my $p = scale_point(2, 3, 2, 2)
p @$p   # (4, 6)
```

scale_to_intervals

`scale_to_intervals` — batch 9 builtin.

```
my $result = scale_to_intervals $input
```

scalept

`scalept` — geometry (extended) builtin. Alias for `scale_point`.

```
my $result = scalept $x
# or in a pipeline:
@list |> map scalept |> p
```

scan

`scan` — functional combinators builtin.

```
my $result = scan $x
# or in a pipeline:
@list |> map scan |> p
```

scan_axis

`scan_axis` — batch 54 builtin.

```
my $result = scan_axis $x
# or in a pipeline:
@list |> map scan_axis |> p
```

scan_left

`scan_left` — list helpers (batch 4) builtin.

```
my $result = scan_left $x
# or in a pipeline:
@list |> map scan_left |> p
```

scanl

`scanl` — haskell list functions builtin.

```
my $result = scanl $x
# or in a pipeline:
@list |> map scanl |> p
```

scanr

`scanr` — haskell list functions builtin.

```
my $result = scanr $x
# or in a pipeline:
@list |> map scanr |> p
```

scard

`scard` — batch 71 builtin.

```
my $result = scard $x
# or in a pipeline:
@list |> map scard |> p
```

scentroid

`spectral_centroid(\@spectrum)` (alias `scentroid`) — computes the spectral centroid (center of mass) of a spectrum. Indicates the 'brightness' of a sound. Returns frequency bin index.

```
my $psd = psd(\@signal)
p scentroid($psd)   # brightness measure
```

schmidt_number_step

`schmidt_number_step` — batch 42 builtin.

```
my $result = schmidt_number_step $x
# or in a pipeline:
@list |> map schmidt_number_step |> p
```

school_choice_match

`school_choice_match` — batch 43 builtin.

```
my $result = school_choice_match $x
# or in a pipeline:
@list |> map school_choice_match |> p
```

schottky_barrier_height

`schottky_barrier_height` — batch 38 builtin.

```
my $result = schottky_barrier_height $x
# or in a pipeline:
@list |> map schottky_barrier_height |> p
```

schouten_tensor_step

`schouten_tensor_step` — batch 39 builtin.

```
my $result = schouten_tensor_step $x
# or in a pipeline:
@list |> map schouten_tensor_step |> p
```

schreier_index

`schreier_index` — batch 37 builtin.

```
my $result = schreier_index $x
# or in a pipeline:
@list |> map schreier_index |> p
```

schrodinger_step_real

`schrodinger_step_real` — batch 32 builtin.

```
my $result = schrodinger_step_real $x
# or in a pipeline:
@list |> map schrodinger_step_real |> p
```

schulze_method_step

`schulze_method_step` — batch 43 builtin.

```
my $result = schulze_method_step $x
# or in a pipeline:
@list |> map schulze_method_step |> p
```

schur_polynomial_eval

`schur_polynomial_eval` — batch 37 builtin.

```
my $result = schur_polynomial_eval $x
# or in a pipeline:
@list |> map schur_polynomial_eval |> p
```

schwarz

`schwarzschild_radius($mass)` (alias `schwarz`) — event horizon radius rs = 2GM/c² of a black hole.

```
p schwarz(1.989e30)   # ~2954 m (Sun as black hole)
p schwarz(5.972e24)   # ~0.009 m (Earth as black hole)
```

schwarzschild_freefall_time

schwarzschild_freefall_time — batch 31 builtin.

```
my $result = schwarzschild_freefall_time $x
# or in a pipeline:
@list |> map schwarzschild_freefall_time |> p
```

schwarzschild_g_rr

schwarzschild_g_rr — batch 31 builtin.

```
my $result = schwarzschild_g_rr $x
# or in a pipeline:
@list |> map schwarzschild_g_rr |> p
```

schwarzschild_g_tt

schwarzschild_g_tt — batch 31 builtin.

```
my $result = schwarzschild_g_tt $x
# or in a pipeline:
@list |> map schwarzschild_g_tt |> p
```

schwarzschild_isco

schwarzschild_isco — batch 31 builtin.

```
my $result = schwarzschild_isco $x
# or in a pipeline:
@list |> map schwarzschild_isco |> p
```

schwarzschild_kruskal_step

schwarzschild_kruskal_step — batch 39 builtin.

```
my $result = schwarzschild_kruskal_step $x
# or in a pipeline:
@list |> map schwarzschild_kruskal_step |> p
```

schwarzschild_radius_kg

schwarzschild_radius_kg — batch 31 builtin.

```
my $result = schwarzschild_radius_kg $x
# or in a pipeline:
@list |> map schwarzschild_radius_kg |> p
```

schwarzschild_radius_m

schwarzschild_radius_m — batch 12 builtin.

```
my $result = schwarzschild_radius_m $x
# or in a pipeline:
@list |> map schwarzschild_radius_m |> p
```

score_test_stat

score_test_stat — batch 36 builtin.

```
my $result = score_test_stat $x
# or in a pipeline:
@list |> map score_test_stat |> p
```

script_name

`script_name` — process / env builtin.

```
my $result = script_name $x
# or in a pipeline:
@list |> map script_name |> p
```

scrypt_round

`scrypt_round` — batch 26 builtin.

```
my $result = scrypt_round $x
# or in a pipeline:
@list |> map scrypt_round |> p
```

sdbm_hash

`sdbm_hash` — batch 26 builtin.

```
my $result = sdbm_hash $x
# or in a pipeline:
@list |> map sdbm_hash |> p
```

sdiff

`sdiff` — batch 71 builtin.

```
my $result = sdiff $x
# or in a pipeline:
@list |> map sdiff |> p
```

sdivs

`sdivs` — math / numeric (uncategorized batch) builtin. Alias for `sum_divisors`.

```
my $result = sdivs $x
# or in a pipeline:
@list |> map sdivs |> p
```

sdns

`sdns` — stress / telemetry extensions builtin. Alias for `stress_dns`.

```
my $result = sdns $x
# or in a pipeline:
@list |> map sdns |> p
```

sdx

`sdx` — extended stdlib builtin. Alias for `soundex`.

```
my $result = sdx $x
# or in a pipeline:
@list |> map sdx |> p
```

sealed_bid_first_price

`sealed_bid_first_price` — batch 43 builtin.

```
my $result = sealed_bid_first_price $x
# or in a pipeline:
@list |> map sealed_bid_first_price |> p
```

sealed_bid_second_price

`sealed_bid_second_price` — batch 43 builtin.

```
my $result = sealed_bid_second_price $x
# or in a pipeline:
@list |> map sealed_bid_second_price |> p
```

searchsorted

`searchsorted` — batch 72 builtin.

```
my $result = searchsorted $x
# or in a pipeline:
@list |> map searchsorted |> p
```

seasonal_decompose

`seasonal_decompose` — batch 78 builtin.

```
my $result = seasonal_decompose $x
# or in a pipeline:
@list |> map seasonal_decompose |> p
```

seasonal_diff

`seasonal_diff` — batches 17-18 builtin.

```
my $result = seasonal_diff $x
# or in a pipeline:
@list |> map seasonal_diff |> p
```

sec

`sec` returns the secant (1/cos) of an angle in radians.

```
p sec(0)  # 1.0
```

sec_2fa_totp_window

`sec_2fa_totp_window` — batch 51 builtin.

```
my $result = sec_2fa_totp_window $x
# or in a pipeline:
@list |> map sec_2fa_totp_window |> p
```

sec_account_lockout_step

`sec_account_lockout_step` — batch 51 builtin.

```
my $result = sec_account_lockout_step $x
# or in a pipeline:
@list |> map sec_account_lockout_step |> p
```

sec_acme_dns_challenge

`sec_acme_dns_challenge` — batch 51 builtin.

```
my $result = sec_acme_dns_challenge $x
# or in a pipeline:
@list |> map sec_acme_dns_challenge |> p
```

sec_aes_keyschedule_step

sec_aes_keyschedule_step — batch 51 builtin.

```
my $result = sec_aes_keyschedule_step $x
# or in a pipeline:
@list |> map sec_aes_keyschedule_step |> p
```

sec_aes_round_step

sec_aes_round_step — batch 51 builtin.

```
my $result = sec_aes_round_step $x
# or in a pipeline:
@list |> map sec_aes_round_step |> p
```

sec_arc_chain_step

sec_arc_chain_step — batch 51 builtin.

```
my $result = sec_arc_chain_step $x
# or in a pipeline:
@list |> map sec_arc_chain_step |> p
```

sec_argon2_block_step

sec_argon2_block_step — batch 51 builtin.

```
my $result = sec_argon2_block_step $x
# or in a pipeline:
@list |> map sec_argon2_block_step |> p
```

sec_argon2_memcost

sec_argon2_memcost — batch 51 builtin.

```
my $result = sec_argon2_memcost $x
# or in a pipeline:
@list |> map sec_argon2_memcost |> p
```

sec_argon2_parallelism

sec_argon2_parallelism — batch 51 builtin.

```
my $result = sec_argon2_parallelism $x
# or in a pipeline:
@list |> map sec_argon2_parallelism |> p
```

sec_argon2_state_advance

sec_argon2_state_advance — batch 51 builtin.

```
my $result = sec_argon2_state_advance $x
# or in a pipeline:
@list |> map sec_argon2_state_advance |> p
```

sec_argon2_timecost

sec_argon2_timecost — batch 51 builtin.

```
my $result = sec_argon2_timecost $x
# or in a pipeline:
@list |> map sec_argon2_timecost |> p
```

sec_balloon_hash_step

`sec_balloon_hash_step` — batch 51 builtin.

```
my $result = sec_balloon_hash_step $x
# or in a pipeline:
@list |> map sec_balloon_hash_step |> p
```

sec_basic_constraints_ca

`sec_basic_constraints_ca` — batch 51 builtin.

```
my $result = sec_basic_constraints_ca $x
# or in a pipeline:
@list |> map sec_basic_constraints_ca |> p
```

sec_bcrypt_cost_factor

`sec_bcrypt_cost_factor` — batch 51 builtin.

```
my $result = sec_bcrypt_cost_factor $x
# or in a pipeline:
@list |> map sec_bcrypt_cost_factor |> p
```

sec_bcrypt_round_step

`sec_bcrypt_round_step` — batch 51 builtin.

```
my $result = sec_bcrypt_round_step $x
# or in a pipeline:
@list |> map sec_bcrypt_round_step |> p
```

sec_blake3_chunk_step

`sec_blake3_chunk_step` — batch 51 builtin.

```
my $result = sec_blake3_chunk_step $x
# or in a pipeline:
@list |> map sec_blake3_chunk_step |> p
```

sec_blowfish_round_step

`sec_blowfish_round_step` — batch 51 builtin.

```
my $result = sec_blowfish_round_step $x
# or in a pipeline:
@list |> map sec_blowfish_round_step |> p
```

sec_brute_force_attempts

`sec_brute_force_attempts` — batch 51 builtin.

```
my $result = sec_brute_force_attempts $x
# or in a pipeline:
@list |> map sec_brute_force_attempts |> p
```

sec_cbc_mac_block_count

`sec_cbc_mac_block_count` — batch 51 builtin.

```
my $result = sec_cbc_mac_block_count $x
# or in a pipeline:
@list |> map sec_cbc_mac_block_count |> p
```

sec_certificate_chain_depth

sec_certificate_chain_depth — batch 51 builtin.

```
my $result = sec_certificate_chain_depth $x
# or in a pipeline:
@list |> map sec_certificate_chain_depth |> p
```

sec_certificate_transparency

sec_certificate_transparency — batch 51 builtin.

```
my $result = sec_certificate_transparency $x
# or in a pipeline:
@list |> map sec_certificate_transparency |> p
```

sec_chacha20_quarterround

sec_chacha20_quarterround — batch 51 builtin.

```
my $result = sec_chacha20_quarterround $x
# or in a pipeline:
@list |> map sec_chacha20_quarterround |> p
```

sec_chachapoly_nonce_check

sec_chachapoly_nonce_check — batch 51 builtin.

```
my $result = sec_chachapoly_nonce_check $x
# or in a pipeline:
@list |> map sec_chachapoly_nonce_check |> p
```

sec_chosen_charset_strength

sec_chosen_charset_strength — batch 51 builtin.

```
my $result = sec_chosen_charset_strength $x
# or in a pipeline:
@list |> map sec_chosen_charset_strength |> p
```

sec_cipher_suite_strength

sec_cipher_suite_strength — batch 51 builtin.

```
my $result = sec_cipher_suite_strength $x
# or in a pipeline:
@list |> map sec_cipher_suite_strength |> p
```

sec_command_injection_score

sec_command_injection_score — batch 51 builtin.

```
my $result = sec_command_injection_score $x
# or in a pipeline:
@list |> map sec_command_injection_score |> p
```

sec_complexity_policy_score

sec_complexity_policy_score — batch 51 builtin.

```
my $result = sec_complexity_policy_score $x
# or in a pipeline:
@list |> map sec_complexity_policy_score |> p
```

sec_cors_origin_match

sec_cors_origin_match — batch 51 builtin.

```
my $result = sec_cors_origin_match $x
# or in a pipeline:
@list |> map sec_cors_origin_match |> p
```

sec_credential_stuffing_score

sec_credential_stuffing_score — batch 51 builtin.

```
my $result = sec_credential_stuffing_score $x
# or in a pipeline:
@list |> map sec_credential_stuffing_score |> p
```

sec_crl_age_seconds

sec_crl_age_seconds — batch 51 builtin.

```
my $result = sec_crl_age_seconds $x
# or in a pipeline:
@list |> map sec_crl_age_seconds |> p
```

sec_csp_directive_match

sec_csp_directive_match — batch 51 builtin.

```
my $result = sec_csp_directive_match $x
# or in a pipeline:
@list |> map sec_csp_directive_match |> p
```

sec_csrf_token_match

sec_csrf_token_match — batch 51 builtin.

```
my $result = sec_csrf_token_match $x
# or in a pipeline:
@list |> map sec_csrf_token_match |> p
```

sec_dane_tlsa_match

sec_dane_tlsa_match — batch 51 builtin.

```
my $result = sec_dane_tlsa_match $x
# or in a pipeline:
@list |> map sec_dane_tlsa_match |> p
```

sec_des_round_step

sec_des_round_step — batch 51 builtin.

```
my $result = sec_des_round_step $x
# or in a pipeline:
@list |> map sec_des_round_step |> p
```

sec_diameter_avp_step

sec_diameter_avp_step — batch 51 builtin.

```
my $result = sec_diameter_avp_step $x
# or in a pipeline:
@list |> map sec_diameter_avp_step |> p
```

sec_diceware_word_index

sec_diceware_word_index — batch 51 builtin.

```
my $result = sec_diceware_word_index $x
# or in a pipeline:
@list |> map sec_diceware_word_index |> p
```

sec_dictionary_attack_check

sec_dictionary_attack_check — batch 51 builtin.

```
my $result = sec_dictionary_attack_check $x
# or in a pipeline:
@list |> map sec_dictionary_attack_check |> p
```

sec_dkim_signature_check

sec_dkim_signature_check — batch 51 builtin.

```
my $result = sec_dkim_signature_check $x
# or in a pipeline:
@list |> map sec_dkim_signature_check |> p
```

sec_dmarc_policy_check

sec_dmarc_policy_check — batch 51 builtin.

```
my $result = sec_dmarc_policy_check $x
# or in a pipeline:
@list |> map sec_dmarc_policy_check |> p
```

sec_dnssec_signature_check

sec_dnssec_signature_check — batch 51 builtin.

```
my $result = sec_dnssec_signature_check $x
# or in a pipeline:
@list |> map sec_dnssec_signature_check |> p
```

sec_ed25519_signature_step

sec_ed25519_signature_step — batch 51 builtin.

```
my $result = sec_ed25519_signature_step $x
# or in a pipeline:
@list |> map sec_ed25519_signature_step |> p
```

sec_ed448_signature_step

sec_ed448_signature_step — batch 51 builtin.

```
my $result = sec_ed448_signature_step $x
# or in a pipeline:
@list |> map sec_ed448_signature_step |> p
```

sec_fido2_assertion_check

sec_fido2_assertion_check — batch 51 builtin.

```
my $result = sec_fido2_assertion_check $x
# or in a pipeline:
@list |> map sec_fido2_assertion_check |> p
```

sec_gcm_iv_unique_check

sec_gcm_iv_unique_check — batch 51 builtin.

```
my $result = sec_gcm_iv_unique_check $x
# or in a pipeline:
@list |> map sec_gcm_iv_unique_check |> p
```

sec_haveibeenpwned_check

sec_haveibeenpwned_check — batch 51 builtin.

```
my $result = sec_haveibeenpwned_check $x
# or in a pipeline:
@list |> map sec_haveibeenpwned_check |> p
```

sec_hotp_counter_step

sec_hotp_counter_step — batch 51 builtin.

```
my $result = sec_hotp_counter_step $x
# or in a pipeline:
@list |> map sec_hotp_counter_step |> p
```

sec_hpkp_pin_match

sec_hpkp_pin_match — batch 51 builtin.

```
my $result = sec_hpkp_pin_match $x
# or in a pipeline:
@list |> map sec_hpkp_pin_match |> p
```

sec_html_escape_check

sec_html_escape_check — batch 51 builtin.

```
my $result = sec_html_escape_check $x
# or in a pipeline:
@list |> map sec_html_escape_check |> p
```

sec_idle_timeout_step

sec_idle_timeout_step — batch 51 builtin.

```
my $result = sec_idle_timeout_step $x
# or in a pipeline:
@list |> map sec_idle_timeout_step |> p
```

sec_idor_check

sec_idor_check — batch 51 builtin.

```
my $result = sec_idor_check $x
# or in a pipeline:
@list |> map sec_idor_check |> p
```

sec_imap_starttls_check

sec_imap_starttls_check — batch 51 builtin.

```
my $result = sec_imap_starttls_check $x
# or in a pipeline:
@list |> map sec_imap_starttls_check |> p
```

sec_jwt_alg_safe

sec_jwt_alg_safe — batch 51 builtin.

```
my $result = sec_jwt_alg_safe $x
# or in a pipeline:
@list |> map sec_jwt_alg_safe |> p
```

sec_jwt_kid_match

sec_jwt_kid_match — batch 51 builtin.

```
my $result = sec_jwt_kid_match $x
# or in a pipeline:
@list |> map sec_jwt_kid_match |> p
```

sec_jwt_signature_verify

sec_jwt_signature_verify — batch 51 builtin.

```
my $result = sec_jwt_signature_verify $x
# or in a pipeline:
@list |> map sec_jwt_signature_verify |> p
```

sec_keccak_round_step

sec_keccak_round_step — batch 51 builtin.

```
my $result = sec_keccak_round_step $x
# or in a pipeline:
@list |> map sec_keccak_round_step |> p
```

sec_kerberos_pac_check

sec_kerberos_pac_check — batch 51 builtin.

```
my $result = sec_kerberos_pac_check $x
# or in a pipeline:
@list |> map sec_kerberos_pac_check |> p
```

sec_kerberos_pre_auth

sec_kerberos_pre_auth — batch 51 builtin.

```
my $result = sec_kerberos_pre_auth $x
# or in a pipeline:
@list |> map sec_kerberos_pre_auth |> p
```

sec_kerberos_ticket_age

sec_kerberos_ticket_age — batch 51 builtin.

```
my $result = sec_kerberos_ticket_age $x
# or in a pipeline:
@list |> map sec_kerberos_ticket_age |> p
```

sec_keystroke_timing_var

sec_keystroke_timing_var — batch 51 builtin.

```
my $result = sec_keystroke_timing_var $x
# or in a pipeline:
@list |> map sec_keystroke_timing_var |> p
```

sec_ldap_bind_step

`sec_ldap_bind_step` — batch 51 builtin.

```
my $result = sec_ldap_bind_step $x
# or in a pipeline:
@list |> map sec_ldap_bind_step |> p
```

sec_login_throttle_step

`sec_login_throttle_step` — batch 51 builtin.

```
my $result = sec_login_throttle_step $x
# or in a pipeline:
@list |> map sec_login_throttle_step |> p
```

sec_oauth2_pkce_step

`sec_oauth2_pkce_step` — batch 51 builtin.

```
my $result = sec_oauth2_pkce_step $x
# or in a pipeline:
@list |> map sec_oauth2_pkce_step |> p
```

sec_oauth2_state_validate

`sec_oauth2_state_validate` — batch 51 builtin.

```
my $result = sec_oauth2_state_validate $x
# or in a pipeline:
@list |> map sec_oauth2_state_validate |> p
```

sec_oauth_nonce_check

`sec_oauth_nonce_check` — batch 51 builtin.

```
my $result = sec_oauth_nonce_check $x
# or in a pipeline:
@list |> map sec_oauth_nonce_check |> p
```

sec_oidc_id_token_age

`sec_oidc_id_token_age` — batch 51 builtin.

```
my $result = sec_oidc_id_token_age $x
# or in a pipeline:
@list |> map sec_oidc_id_token_age |> p
```

sec_p384_curve_step

`sec_p384_curve_step` — batch 51 builtin.

```
my $result = sec_p384_curve_step $x
# or in a pipeline:
@list |> map sec_p384_curve_step |> p
```

sec_passphrase_entropy

`sec_passphrase_entropy` — batch 51 builtin.

```
my $result = sec_passphrase_entropy $x
# or in a pipeline:
@list |> map sec_passphrase_entropy |> p
```

sec_password_history_check

sec_password_history_check — batch 51 builtin.

```
my $result = sec_password_history_check $x
# or in a pipeline:
@list |> map sec_password_history_check |> p
```

sec_password_strength_zxcvbn

sec_password_strength_zxcvbn — batch 51 builtin.

```
my $result = sec_password_strength_zxcvbn $x
# or in a pipeline:
@list |> map sec_password_strength_zxcvbn |> p
```

sec_path_traversal_detect

sec_path_traversal_detect — batch 51 builtin.

```
my $result = sec_path_traversal_detect $x
# or in a pipeline:
@list |> map sec_path_traversal_detect |> p
```

sec_pbkdf2_iter

sec_pbkdf2_iter — batch 51 builtin.

```
my $result = sec_pbkdf2_iter $x
# or in a pipeline:
@list |> map sec_pbkdf2_iter |> p
```

sec_pinning_compare

sec_pinning_compare — batch 51 builtin.

```
my $result = sec_pinning_compare $x
# or in a pipeline:
@list |> map sec_pinning_compare |> p
```

sec_pki_path_validate

sec_pki_path_validate — batch 51 builtin.

```
my $result = sec_pki_path_validate $x
# or in a pipeline:
@list |> map sec_pki_path_validate |> p
```

sec_pop3_security_step

sec_pop3_security_step — batch 51 builtin.

```
my $result = sec_pop3_security_step $x
# or in a pipeline:
@list |> map sec_pop3_security_step |> p
```

sec_radius_auth_step

sec_radius_auth_step — batch 51 builtin.

```
my $result = sec_radius_auth_step $x
# or in a pipeline:
@list |> map sec_radius_auth_step |> p
```

sec_revocation_ocsp_check

sec_revocation_ocsp_check — batch 51 builtin.

```
my $result = sec_revocation_ocsp_check $x
# or in a pipeline:
@list |> map sec_revocation_ocsp_check |> p
```

sec_saml_assertion_age

sec_saml_assertion_age — batch 51 builtin.

```
my $result = sec_saml_assertion_age $x
# or in a pipeline:
@list |> map sec_saml_assertion_age |> p
```

sec_san_match_count

sec_san_match_count — batch 51 builtin.

```
my $result = sec_san_match_count $x
# or in a pipeline:
@list |> map sec_san_match_count |> p
```

sec_scrypt_n_param

sec_scrypt_n_param — batch 51 builtin.

```
my $result = sec_scrypt_n_param $x
# or in a pipeline:
@list |> map sec_scrypt_n_param |> p
```

sec_scrypt_p_param

sec_scrypt_p_param — batch 51 builtin.

```
my $result = sec_scrypt_p_param $x
# or in a pipeline:
@list |> map sec_scrypt_p_param |> p
```

sec_scrypt_r_param

sec_scrypt_r_param — batch 51 builtin.

```
my $result = sec_scrypt_r_param $x
# or in a pipeline:
@list |> map sec_scrypt_r_param |> p
```

sec_secp256k1_step

sec_secp256k1_step — batch 51 builtin.

```
my $result = sec_secp256k1_step $x
# or in a pipeline:
@list |> map sec_secp256k1_step |> p
```

sec_serpent_round_step

sec_serpent_round_step — batch 51 builtin.

```
my $result = sec_serpent_round_step $x
# or in a pipeline:
@list |> map sec_serpent_round_step |> p
```

sec_session_lifetime

`sec_session_lifetime` — batch 51 builtin.

```
my $result = sec_session_lifetime $x
# or in a pipeline:
@list |> map sec_session_lifetime |> p
```

sec_sha3_padding_step

`sec_sha3_padding_step` — batch 51 builtin.

```
my $result = sec_sha3_padding_step $x
# or in a pipeline:
@list |> map sec_sha3_padding_step |> p
```

sec_smtp_ssl_check

`sec_smtp_ssl_check` — batch 51 builtin.

```
my $result = sec_smtp_ssl_check $x
# or in a pipeline:
@list |> map sec_smtp_ssl_check |> p
```

sec_spf_pass_check

`sec_spf_pass_check` — batch 51 builtin.

```
my $result = sec_spf_pass_check $x
# or in a pipeline:
@list |> map sec_spf_pass_check |> p
```

sec_sqli_pattern_score

`sec_sqli_pattern_score` — batch 51 builtin.

```
my $result = sec_sqli_pattern_score $x
# or in a pipeline:
@list |> map sec_sqli_pattern_score |> p
```

sec_ssl3_disabled_check

`sec_ssl3_disabled_check` — batch 51 builtin.

```
my $result = sec_ssl3_disabled_check $x
# or in a pipeline:
@list |> map sec_ssl3_disabled_check |> p
```

sec_tls11_deprecation_check

`sec_tls11_deprecation_check` — batch 51 builtin.

```
my $result = sec_tls11_deprecation_check $x
# or in a pipeline:
@list |> map sec_tls11_deprecation_check |> p
```

sec_tls12_handshake_step

`sec_tls12_handshake_step` — batch 51 builtin.

```
my $result = sec_tls12_handshake_step $x
# or in a pipeline:
@list |> map sec_tls12_handshake_step |> p
```

sec_tls13_handshake_step

sec_tls13_handshake_step — batch 51 builtin.

```
my $result = sec_tls13_handshake_step $x
# or in a pipeline:
@list |> map sec_tls13_handshake_step |> p
```

sec_tls_alert_severity

sec_tls_alert_severity — batch 51 builtin.

```
my $result = sec_tls_alert_severity $x
# or in a pipeline:
@list |> map sec_tls_alert_severity |> p
```

sec_totp_drift_check

sec_totp_drift_check — batch 51 builtin.

```
my $result = sec_totp_drift_check $x
# or in a pipeline:
@list |> map sec_totp_drift_check |> p
```

sec_twofish_round_step

sec_twofish_round_step — batch 51 builtin.

```
my $result = sec_twofish_round_step $x
# or in a pipeline:
@list |> map sec_twofish_round_step |> p
```

sec_url_safe_encode_check

sec_url_safe_encode_check — batch 51 builtin.

```
my $result = sec_url_safe_encode_check $x
# or in a pipeline:
@list |> map sec_url_safe_encode_check |> p
```

sec_webauthn_attestation_check

sec_webauthn_attestation_check — batch 51 builtin.

```
my $result = sec_webauthn_attestation_check $x
# or in a pipeline:
@list |> map sec_webauthn_attestation_check |> p
```

sec_x25519_clamping_step

sec_x25519_clamping_step — batch 51 builtin.

```
my $result = sec_x25519_clamping_step $x
# or in a pipeline:
@list |> map sec_x25519_clamping_step |> p
```

sec_x509_subject_match

sec_x509_subject_match — batch 51 builtin.

```
my $result = sec_x509_subject_match $x
# or in a pipeline:
@list |> map sec_x509_subject_match |> p
```

sec_xkcd_passphrase_score

`sec_xkcd_passphrase_score` — batch 51 builtin.

```
my $result = sec_xkcd_passphrase_score $x
# or in a pipeline:
@list |> map sec_xkcd_passphrase_score |> p
```

sec_xss_filter_score

`sec_xss_filter_score` — batch 51 builtin.

```
my $result = sec_xss_filter_score $x
# or in a pipeline:
@list |> map sec_xss_filter_score |> p
```

sec_xxe_dtd_check

`sec_xxe_dtd_check` — batch 51 builtin.

```
my $result = sec_xxe_dtd_check $x
# or in a pipeline:
@list |> map sec_xxe_dtd_check |> p
```

sec_xxe_pattern_score

`sec_xxe_pattern_score` — batch 51 builtin.

```
my $result = sec_xxe_pattern_score $x
# or in a pipeline:
@list |> map sec_xxe_pattern_score |> p
```

sec_yescrypt_step

`sec_yescrypt_step` — batch 51 builtin.

```
my $result = sec_yescrypt_step $x
# or in a pipeline:
@list |> map sec_yescrypt_step |> p
```

sec_yubikey_otp_check

`sec_yubikey_otp_check` — batch 51 builtin.

```
my $result = sec_yubikey_otp_check $x
# or in a pipeline:
@list |> map sec_yubikey_otp_check |> p
```

secant

`secant` — batch 19 builtin. Alias for `secant_root`.

```
my $result = secant $x
# or in a pipeline:
@list |> map secant |> p
```

secant_root

`secant_root` — batch 19 builtin.

```
my $result = secant_root $x
# or in a pipeline:
@list |> map secant_root |> p
```

second

`second` — algebraic match builtin.

```
my $result = second $x
# or in a pipeline:
@list |> map second |> p
```

second_arg

`second_arg` — functional primitives builtin.

```
my $result = second_arg $x
# or in a pipeline:
@list |> map second_arg |> p
```

second_elem

`second_elem` — list helpers (batch 4) builtin.

```
my $result = second_elem $x
# or in a pipeline:
@list |> map second_elem |> p
```

second_order_concentration

`second_order_concentration` — batch 21 builtin.

```
my $result = second_order_concentration $x
# or in a pipeline:
@list |> map second_order_concentration |> p
```

second_order_half_life

`second_order_half_life` — batch 21 builtin.

```
my $result = second_order_half_life $x
# or in a pipeline:
@list |> map second_order_half_life |> p
```

seconds_to_days

`seconds_to_days` — unit conversions builtin.

```
my $result = seconds_to_days $input
```

seconds_to_hours

`seconds_to_hours` — unit conversions builtin.

```
my $result = seconds_to_hours $input
```

seconds_to_minutes

`seconds_to_minutes` — unit conversions builtin.

```
my $result = seconds_to_minutes $input
```

secp256k1_y_recover

`secp256k1_y_recover` — batches 17-18 builtin.

```
my $result = secp256k1_y_recover $x
# or in a pipeline:
@list |> map secp256k1_y_recover |> p
```

sectarea

`sectarea` — geometry (extended) builtin. Alias for `sector_area`.

```
my $result = sectarea $x
# or in a pipeline:
@list |> map sectarea |> p
```

sectional_curvature_two_plane

`sectional_curvature_two_plane` — batch 39 builtin.

```
my $result = sectional_curvature_two_plane $x
# or in a pipeline:
@list |> map sectional_curvature_two_plane |> p
```

sector_area

`sector_area($radius, $theta)` — computes the area of a circular sector. Theta is the central angle in radians. Returns `0.5 * r² * theta`.

```
p sector_area(10, 3.14159)   # ~157 (half circle)
p sector_area(5, 1.57)       # ~19.6
```

seg_sum

`seg_sum` — batch 10 builtin. Alias for `segment_tree_sum`.

```
my $result = seg_sum $x
# or in a pipeline:
@list |> map seg_sum |> p
```

segment_distance_point

`segment_distance_point` — batch 58 builtin.

```
my $result = segment_distance_point $x
# or in a pipeline:
@list |> map segment_distance_point |> p
```

segment_intersection

`segment_intersection` — batch 58 builtin.

```
my $result = segment_intersection $x
# or in a pipeline:
@list |> map segment_intersection |> p
```

segment_length

`segment_length` — batches 17-18 builtin.

```
my $result = segment_length $x
# or in a pipeline:
@list |> map segment_length |> p
```

segment_tree_sum

`segment_tree_sum` — batch 10 builtin.

```
my $result = segment_tree_sum $x
# or in a pipeline:
@list |> map segment_tree_sum |> p
```

segments_parallel_q

`segments_parallel_q` — batches 17-18 builtin.

```
my $result = segments_parallel_q $x
# or in a pipeline:
@list |> map segments_parallel_q |> p
```

segments_perpendicular_q

`segments_perpendicular_q` — batches 17-18 builtin.

```
my $result = segments_perpendicular_q $x
# or in a pipeline:
@list |> map segments_perpendicular_q |> p
```

sei_resistance_growth

`sei_resistance_growth` — batch 38 builtin.

```
my $result = sei_resistance_growth $x
# or in a pipeline:
@list |> map sei_resistance_growth |> p
```

seiberg_witten_value

`seiberg_witten_value` — batch 37 builtin.

```
my $result = seiberg_witten_value $x
# or in a pipeline:
@list |> map seiberg_witten_value |> p
```

seifert_form_2x2

`seifert_form_2x2` — batch 37 builtin.

```
my $result = seifert_form_2x2 $x
# or in a pipeline:
@list |> map seifert_form_2x2 |> p
```

seifert_genus_lower

`seifert_genus_lower` — batch 37 builtin.

```
my $result = seifert_genus_lower $x
# or in a pipeline:
@list |> map seifert_genus_lower |> p
```

seir_step

`seir_step` — batch 22 builtin.

```
my $result = seir_step $x
# or in a pipeline:
@list |> map seir_step |> p
```

seird_step

`seird_step` — batch 22 builtin.

```
my $result = seird_step $x
# or in a pipeline:
@list |> map seird_step |> p
```

seirs_step

`seirs_step` — batch 61 builtin.

```
my $result = seirs_step $x
# or in a pipeline:
@list |> map seirs_step |> p
```

select_period_step

`select_period_step` — batch 60 builtin.

```
my $result = select_period_step $x
# or in a pipeline:
@list |> map select_period_step |> p
```

select_ultimate

`select_ultimate` — batch 60 builtin.

```
my $result = select_ultimate $x
# or in a pipeline:
@list |> map select_ultimate |> p
```

selection_step

`selection_step` — batch 22 builtin.

```
my $result = selection_step $x
# or in a pipeline:
@list |> map selection_step |> p
```

selu

`selu` applies the Scaled ELU with fixed lambda=1.0507, alpha=1.6733 for self-normalizing networks.

```
p selu(1)    # 1.0507
p selu(-1)   # -1.1113
```

semctl

`semctl` — sysv ipc builtin.

```
my $result = semctl $x
# or in a pipeline:
@list |> map semctl |> p
```

semget

`semget` — sysv ipc builtin.

```
my $result = semget $x
# or in a pipeline:
@list |> map semget |> p
```

semi_major_axis

`semi_major_axis` — batches 14-16 builtin.

```
my $result = semi_major_axis $x
# or in a pipeline:
@list |> map semi_major_axis |> p
```

semop

`semop` — sysv ipc builtin.

```
my $result = semop $x
# or in a pipeline:
@list |> map semop |> p
```

sentence_case

`sentence_case` — string helpers (batch 4) builtin.

```
my $result = sentence_case $x
# or in a pipeline:
@list |> map sentence_case |> p
```

seq

`seq` — algebraic match builtin.

```
my $result = seq $x
# or in a pipeline:
@list |> map seq |> p
```

seq_fn

`seq_fn` — generate a numeric sequence from, to, by. Like R's seq().

```
my @s = @{seq_fn(1, 10, 2)}  # [1,3,5,7,9]
my @r = @{seq_fn(5, 1, -1)}  # [5,4,3,2,1]
```

seq_logo_information

`seq_logo_information` — batch 33 builtin.

```
my $result = seq_logo_information $x
# or in a pipeline:
@list |> map seq_logo_information |> p
```

sequence_shannon_info

`sequence_shannon_info` — batch 33 builtin.

```
my $result = sequence_shannon_info $x
# or in a pipeline:
@list |> map sequence_shannon_info |> p
```

sequent_left_intro

`sequent_left_intro` — batch 67 builtin.

```
my $result = sequent_left_intro $x
# or in a pipeline:
@list |> map sequent_left_intro |> p
```

sequent_right_intro

`sequent_right_intro` — batch 67 builtin.

```
my $result = sequent_right_intro $x
# or in a pipeline:
@list |> map sequent_right_intro |> p
```

sequential_eq_check

`sequential_eq_check` — batch 43 builtin.

```
my $result = sequential_eq_check $x
# or in a pipeline:
@list |> map sequential_eq_check |> p
```

servo_position_velocity

`servo_position_velocity` — batch 59 builtin.

```
my $result = servo_position_velocity $x
# or in a pipeline:
@list |> map servo_position_velocity |> p
```

servo_torque_step

`servo_torque_step` — batch 59 builtin.

```
my $result = servo_torque_step $x
# or in a pipeline:
@list |> map servo_torque_step |> p
```

set_at

`set_at` — batch 9 builtin. Alias for `replace_at_index`.

```
my $result = set_at $x
# or in a pipeline:
@list |> map set_at |> p
```

set_cover_greedy_step

`set_cover_greedy_step` — batch 41 builtin.

```
my $result = set_cover_greedy_step $x
# or in a pipeline:
@list |> map set_cover_greedy_step |> p
```

set_cover_lp_round

`set_cover_lp_round` — batch 41 builtin.

```
my $result = set_cover_lp_round $x
# or in a pipeline:
@list |> map set_cover_lp_round |> p
```

set_subname

`set_subname` — list / aggregate builtin.

```
my $result = set_subname $x
# or in a pipeline:
@list |> map set_subname |> p
```

setex

`setex` — batch 71 builtin.

```
my $result = setex $x
# or in a pipeline:
@list |> map setex |> p
```

setgrent

`setgrent` — posix metadata builtin.

```
my $result = setgrent $x
# or in a pipeline:
@list |> map setgrent |> p
```

sethostent

`sethostent` — posix metadata builtin.

```
my $result = sethostent $x
# or in a pipeline:
@list |> map sethostent |> p
```

setnetent

`setnetent` — posix metadata builtin.

```
my $result = setnetent $x
# or in a pipeline:
@list |> map setnetent |> p
```

setnx

`setnx` — batch 71 builtin.

```
my $result = setnx $x
# or in a pipeline:
@list |> map setnx |> p
```

setprotoent

`setprotoent` — posix metadata builtin.

```
my $result = setprotoent $x
# or in a pipeline:
@list |> map setprotoent |> p
```

setpwent

`setpwent` — posix metadata builtin.

```
my $result = setpwent $x
# or in a pipeline:
@list |> map setpwent |> p
```

setrange

`setrange` — batch 71 builtin.

```
my $result = setrange $x
# or in a pipeline:
@list |> map setrange |> p
```

setservent

`setservent` — posix metadata builtin.

```
my $result = setservent $x
# or in a pipeline:
@list |> map setservent |> p
```

seven_smooth_q

`seven_smooth_q` — batch 10 builtin.

```
my $result = seven_smooth_q $x
# or in a pipeline:
@list |> map seven_smooth_q |> p
```

sfork

`sfork` — stress / telemetry extensions builtin. Alias for `stress_fork`.

```
my $result = sfork $x
# or in a pipeline:
@list |> map sfork |> p
```

sgemm

`sgemm` — batch 66 builtin.

```
my $result = sgemm $x
# or in a pipeline:
@list |> map sgemm |> p
```

sgemv

`sgemv` — batch 66 builtin.

```
my $result = sgemv $x
# or in a pipeline:
@list |> map sgemv |> p
```

sgnum

`sgnum` — extended stdlib builtin. Alias for `signum_of`.

```
my $result = sgnum $x
# or in a pipeline:
@list |> map sgnum |> p
```

sgp4_propagate_step

`sgp4_propagate_step` — batch 55 builtin.

```
my $result = sgp4_propagate_step $x
# or in a pipeline:
@list |> map sgp4_propagate_step |> p
```

shake_map_pga

`shake_map_pga` — batch 65 builtin.

```
my $result = shake_map_pga $x
# or in a pipeline:
@list |> map shake_map_pga |> p
```

shanks_transform

`shanks_transform` — batches 14-16 builtin.

```
my $result = shanks_transform $x
# or in a pipeline:
@list |> map shanks_transform |> p
```

shannon_fano_elias_code

`shannon_fano_elias_code` — batch 40 builtin.

```
my $result = shannon_fano_elias_code $x
# or in a pipeline:
@list |> map shannon_fano_elias_code |> p
```

shape_operator_eig

`shape_operator_eig` — batch 39 builtin.

```
my $result = shape_operator_eig $x
# or in a pipeline:
@list |> map shape_operator_eig |> p
```

shapiro

`shapiro_test` (alias `shapiro`) — Shapiro-Wilk normality test. Returns W statistic (close to 1 = normal). Like R's shapiro.test().

```
p shapiro([rnorm(100)])  # W ≈ 0.99 for normal data
p shapiro([1,1,1,2,10]) # W < 0.9 for non-normal
```

shapiro_delay

`shapiro_delay` — batch 31 builtin.

```
my $result = shapiro_delay $x
# or in a pipeline:
@list |> map shapiro_delay |> p
```

shapiro_delay_step

`shapiro_delay_step` — batch 39 builtin.

```
my $result = shapiro_delay_step $x
# or in a pipeline:
@list |> map shapiro_delay_step |> p
```

shapiro_wilk

`shapiro_wilk` — batch 53 builtin.

```
my $result = shapiro_wilk $x
# or in a pipeline:
@list |> map shapiro_wilk |> p
```

shapley_value_two_step

`shapley_value_two_step` — batch 43 builtin.

```
my $result = shapley_value_two_step $x
# or in a pipeline:
@list |> map shapley_value_two_step |> p
```

sharpe

`sharpe_ratio(\@returns, $risk_free_rate)` (alias `sharpe`) — computes the Sharpe ratio measuring risk-adjusted returns. Higher is better. Returns (mean_return - rf) / stddev.

```
my @returns = (0.05, -0.02, 0.08, 0.03)
p sharpe(\@returns, 0.02)   # risk-adjusted performance
```

sharpe_annualized

`sharpe_annualized` — batch 20 builtin.

```
my $result = sharpe_annualized $x
# or in a pipeline:
@list |> map sharpe_annualized |> p
```

sharpen_kernel

`sharpen_kernel` — batches 14-16 builtin.

```
my $result = sharpen_kernel $x
# or in a pipeline:
@list |> map sharpen_kernel |> p
```

shear_tensor_step

`shear_tensor_step` — batch 39 builtin.

```
my $result = shear_tensor_step $x
# or in a pipeline:
@list |> map shear_tensor_step |> p
```

shepherd_voltage_step

`shepherd_voltage_step` — batch 38 builtin.

```
my $result = shepherd_voltage_step $x
# or in a pipeline:
@list |> map shepherd_voltage_step |> p
```

sherwood_number_step

`sherwood_number_step` — batch 42 builtin.

```
my $result = sherwood_number_step $x
# or in a pipeline:
@list |> map sherwood_number_step |> p
```

shi_tomasi_corners

`shi_tomasi_corners` — batch 77 builtin.

```
my $result = shi_tomasi_corners $x
# or in a pipeline:
@list |> map shi_tomasi_corners |> p
```

shift_left

`shift_left` — bit ops builtin.

```
my $result = shift_left $x
# or in a pipeline:
@list |> map shift_left |> p
```

shift_right

`shift_right` — bit ops builtin.

```
my $result = shift_right $x
# or in a pipeline:
@list |> map shift_right |> p
```

shl

`shl` — bit ops builtin. Alias for `shift_left`.

```
my $result = shl $x
# or in a pipeline:
@list |> map shl |> p
```

shmctl

`shmctl` — sysv ipc builtin.

```
my $result = shmctl $x
# or in a pipeline:
@list |> map shmctl |> p
```

shmget

`shmget` — sysv ipc builtin.

```
my $result = shmget $x
# or in a pipeline:
@list |> map shmget |> p
```

shmread

`shmread` — sysv ipc builtin.

```
my $result = shmread $x
# or in a pipeline:
@list |> map shmread |> p
```

shmwrite

`shmwrite` — sysv ipc builtin.

```
my $result = shmwrite $x
# or in a pipeline:
@list |> map shmwrite |> p
```

shoelace_area

`shoelace_area` — batch 28 builtin.

```
my $result = shoelace_area $x
# or in a pipeline:
@list |> map shoelace_area |> p
```

shooting_pct

`shooting_pct` — batch 56 builtin.

```
my $result = shooting_pct $x
# or in a pipeline:
@list |> map shooting_pct |> p
```

shor_period

`shor_period` — batch 80 builtin.

```
my $result = shor_period $x
# or in a pipeline:
@list |> map shor_period |> p
```

shor_period_prob

`shor_period_prob` — batch 35 builtin.

```
my $result = shor_period_prob $x
# or in a pipeline:
@list |> map shor_period_prob |> p
```

short_id

`short_id` — id helpers builtin.

```
my $result = short_id $x
# or in a pipeline:
@list |> map short_id |> p
```

shorten

`shorten` — trivial string ops builtin. Alias for `truncate_at`.

```
my $result = shorten $x
# or in a pipeline:
@list |> map shorten |> p
```

shortest

`shortest` — collection (batch 2) builtin.

```
my $result = shortest $x
# or in a pipeline:
@list |> map shortest |> p
```

shortest_path_bfs

`shortest_path_bfs` — extended stdlib builtin.

```
my $result = shortest_path_bfs $x
# or in a pipeline:
@list |> map shortest_path_bfs |> p
```

shr

`shr` — bit ops builtin. Alias for `shift_right`.

```
my $result = shr $x
# or in a pipeline:
@list |> map shr |> p
```

shuf

`shuf` — extended stdlib builtin. Alias for `shuffle_arr`.

```
my $result = shuf $x
# or in a pipeline:
@list |> map shuf |> p
```

shuffle_arr

`shuffle_arr` — extended stdlib builtin.

```
my $result = shuffle_arr $x
# or in a pipeline:
@list |> map shuffle_arr |> p
```

shuffle_chars

`shuffle_chars` — file stat / path builtin.

```
my $result = shuffle_chars $x
# or in a pipeline:
@list |> map shuffle_chars |> p
```

shuffled

`shuffled` — functional / iterator builtin.

```
my $result = shuffled $x
# or in a pipeline:
@list |> map shuffled |> p
```

shunting_yard_step

`shunting_yard_step` — batch 68 builtin.

```
my $result = shunting_yard_step $x
# or in a pipeline:
@list |> map shunting_yard_step |> p
```

sidereal_day_period

`sidereal_day_period` — batch 9 builtin.

```
my $result = sidereal_day_period $x
# or in a pipeline:
@list |> map sidereal_day_period |> p
```

siera

`siera` — batch 56 builtin.

```
my $result = siera $x
# or in a pipeline:
@list |> map siera |> p
```

sierp

`sierp` — algorithms / puzzles builtin. Alias for `sierpinski`.

```
my $result = sierp $x
# or in a pipeline:
@list |> map sierp |> p
```

sierpinski

`sierpinski` — algorithms / puzzles builtin.

```
my $result = sierpinski $x
# or in a pipeline:
@list |> map sierpinski |> p
```

sieve

`sieve` — number theory / primes builtin. Alias for `primes_up_to`.

```
my $result = sieve $x
# or in a pipeline:
@list |> map sieve |> p
```

sift_keypoints

`sift_keypoints` — batch 77 builtin.

```
my $result = sift_keypoints $x
# or in a pipeline:
@list |> map sift_keypoints |> p
```

sigma8_default

`sigma8_default` — batch 31 builtin.

```
my $result = sigma8_default $x
# or in a pipeline:
@list |> map sigma8_default |> p
```

sigma_crit

`sigma_crit` — batch 31 builtin.

```
my $result = sigma_crit $x
# or in a pipeline:
@list |> map sigma_crit |> p
```

sigma_sb

`sigma_sb` — physics constants builtin. Alias for `stefan_boltzmann_constant`.

```
my $result = sigma_sb $x
# or in a pipeline:
@list |> map sigma_sb |> p
```

sigmoid

`sigmoid` — math functions builtin.

```
my $result = sigmoid $x
# or in a pipeline:
@list |> map sigmoid |> p
```

sigmoid_grad

`sigmoid_grad` — batch 27 builtin.

```
my $result = sigmoid_grad $x
# or in a pipeline:
@list |> map sigmoid_grad |> p
```

sign

`sign` — trivial numeric / predicate builtins builtin.

```
my $result = sign $x
# or in a pipeline:
@list |> map sign |> p
```

signal_name

`signal_name` — more process / system builtin.

```
my $result = signal_name $x
# or in a pipeline:
@list |> map signal_name |> p
```

signum

`signum` — math functions builtin.

```
my $result = signum $x
# or in a pipeline:
@list |> map signum |> p
```

signum_of

`signum_of` — extended stdlib builtin.

```
my $result = signum_of $x
# or in a pipeline:
@list |> map signum_of |> p
```

silicate_classify

`silicate_classify` — batch 65 builtin.

```
my $result = silicate_classify $x
# or in a pipeline:
@list |> map silicate_classify |> p
```

silu

`silu` (alias `swish`) applies SiLU/Swish: x·sigmoid(x). Smooth approximation to ReLU.

```
p silu(1)    # 0.7311
p swish(-2)  # -0.2384
```

silver

`silver_ratio` (alias `silver`) — δS = 1 + □2 □ 2.414. Related to Pell numbers.

```
p silver   # 2.4142135623730951
```

sim

`sim` — extended stdlib builtin. Alias for `similarity`.

```
my $result = sim $x
# or in a pipeline:
@list |> map sim |> p
```

similarity

`similarity` — extended stdlib builtin.

```
my $result = similarity $x
# or in a pipeline:
@list |> map similarity |> p
```

similarity_function_phi

`similarity_function_phi` — batch 42 builtin.

```
my $result = similarity_function_phi $x
# or in a pipeline:
@list |> map similarity_function_phi |> p
```

simon_round

`simon_round` — batches 14-16 builtin.

```
my $result = simon_round $x
# or in a pipeline:
@list |> map simon_round |> p
```

simple_interest

`simple_interest` — physics formulas builtin.

```
my $result = simple_interest $x
# or in a pipeline:
@list |> map simple_interest |> p
```

simple_interest_compute

`simple_interest_compute` — batch 13 builtin.

```
my $result = simple_interest_compute $x
# or in a pipeline:
@list |> map simple_interest_compute |> p
```

simplex_volume_3d

`simplex_volume_3d` — batch 28 builtin.

```
my $result = simplex_volume_3d $x
# or in a pipeline:
@list |> map simplex_volume_3d |> p
```

simplex_volume_n

`simplex_volume_n` — batch 37 builtin.

```
my $result = simplex_volume_n $x
# or in a pipeline:
@list |> map simplex_volume_n |> p
```

simplicial_volume

`simplicial_volume` — batch 37 builtin.

```
my $result = simplicial_volume $x
# or in a pipeline:
@list |> map simplicial_volume |> p
```

simply_typed_check

`simply_typed_check` — batch 67 builtin.

```
my $result = simply_typed_check $x
# or in a pipeline:
@list |> map simply_typed_check |> p
```

simps

`simpson` (alias `simps`) integrates evenly-spaced samples using Simpson's rule. More accurate than trapz for smooth functions.

```
my @y = map { sin(_ * 0.01) } 0:314
p simps(\@y, 0.01)  # ≈ 2.0
```

simpson_rule

`simpson_rule` — batch 72 builtin.

```
my $result = simpson_rule $x
# or in a pipeline:
@list |> map simpson_rule |> p
```

simrank

`simrank` — batch 75 builtin.

```
my $result = simrank $x
# or in a pipeline:
@list |> map simrank |> p
```

simulated_annealing_step

`simulated_annealing_step` — batch 41 builtin.

```
my $result = simulated_annealing_step $x
# or in a pipeline:
@list |> map simulated_annealing_step |> p
```

sinc

`sinc` returns the unnormalized sinc function: sin(x)/x, with sinc(0)=1.

```
p sinc(0)       # 1.0
p sinc(3.14159) # ≈ 0
```

sine_integral_si

`sine_integral_si` — batch 29 builtin.

```
my $result = sine_integral_si $x
# or in a pipeline:
@list |> map sine_integral_si |> p
```

single_quote

`single_quote` — string quote / escape builtin.

```
my $result = single_quote $x
# or in a pipeline:
@list |> map single_quote |> p
```

single_slit_min

`single_slit_min` — batch 23 builtin.

```
my $result = single_slit_min $x
# or in a pipeline:
@list |> map single_slit_min |> p
```

single_transferable_vote

`single_transferable_vote` — batch 43 builtin.

```
my $result = single_transferable_vote $x
# or in a pipeline:
@list |> map single_transferable_vote |> p
```

singularity_check_2link

`singularity_check_2link` — batch 59 builtin.

```
my $result = singularity_check_2link $x
# or in a pipeline:
@list |> map singularity_check_2link |> p
```

singularize

`singularize` — batch 9 builtin. Alias for `singularize_simple`.

```
my $result = singularize $x
# or in a pipeline:
@list |> map singularize |> p
```

singularize_simple

singularize_simple — batch 9 builtin.

```
my $result = singularize_simple $x
# or in a pipeline:
@list |> map singularize_simple |> p
```

sinh

sinh — trig / math (batch 2) builtin.

```
my $result = sinh $x
# or in a pipeline:
@list |> map sinh |> p
```

sinhc

sinhc — batch 29 builtin.

```
my $result = sinhc $x
# or in a pipeline:
@list |> map sinhc |> p
```

sinkhorn_iteration_step

sinkhorn_iteration_step — batch 40 builtin.

```
my $result = sinkhorn_iteration_step $x
# or in a pipeline:
@list |> map sinkhorn_iteration_step |> p
```

sinter

sinter — batch 71 builtin.

```
my $result = sinter $x
# or in a pipeline:
@list |> map sinter |> p
```

sinusoidal_project

sinusoidal_project — batches 14-16 builtin.

```
my $result = sinusoidal_project $x
# or in a pipeline:
@list |> map sinusoidal_project |> p
```

siphash24

siphash24 — batch 26 builtin.

```
my $result = siphash24 $x
# or in a pipeline:
@list |> map siphash24 |> p
```

siphash_keyed

siphash_keyed computes SipHash-2-4 with a custom 128-bit key (two 64-bit integers). The key should be random per application to prevent collision attacks.

```
p siphash_keyed("data", 0x123456789, 0xabcdef012)  # keyed hash
```

sirs_step

`sirs_step` — batch 61 builtin.

```
my $result = sirs_step $x
# or in a pipeline:
@list |> map sirs_step |> p
```

sis_step

`sis_step` — batch 22 builtin.

```
my $result = sis_step $x
# or in a pipeline:
@list |> map sis_step |> p
```

sismember

`sismember` — batch 71 builtin.

```
my $result = sismember $x
# or in a pipeline:
@list |> map sismember |> p
```

size

`size` returns the byte size of a file on disk — equivalent to Perl's `-s FILE` file test. With no arguments, it operates on `_`; with one argument, it stats the given path; with multiple arguments (or a flattened list), it returns an array of sizes. Paths that can't be stat'd return `undef`. This is a stryke extension that makes pipelines over filenames concise.

```
p size "Cargo.toml"                  # 2013
f |> map +{ _ => size } |> tj |> p   # [{name => bytes}, ...]
f |> filter { size > 1024 } |> e p    # files larger than 1 KiB
```

sjson

`sjson` — stress / telemetry extensions builtin. Alias for `stress_json`.

```
my $result = sjson $x
# or in a pipeline:
@list |> map sjson |> p
```

sk_adjusted_rand

`sk_adjusted_rand` — batch 79 builtin.

```
my $result = sk_adjusted_rand $x
# or in a pipeline:
@list |> map sk_adjusted_rand |> p
```

sk_agglomerative_fit

`sk_agglomerative_fit` — batch 79 builtin.

```
my $result = sk_agglomerative_fit $x
# or in a pipeline:
@list |> map sk_agglomerative_fit |> p
```

sk_bayes_search

`sk_bayes_search` — batch 79 builtin.

```
my $result = sk_bayes_search $x
# or in a pipeline:
@list |> map sk_bayes_search |> p
```

sk_calinski_harabasz

`sk_calinski_harabasz` — batch 79 builtin.

```
my $result = sk_calinski_harabasz $x
# or in a pipeline:
@list |> map sk_calinski_harabasz |> p
```

sk_count_vectorize

`sk_count_vectorize` — batch 79 builtin.

```
my $result = sk_count_vectorize $x
# or in a pipeline:
@list |> map sk_count_vectorize |> p
```

sk_cross_val_score

`sk_cross_val_score` — batch 79 builtin.

```
my $result = sk_cross_val_score $x
# or in a pipeline:
@list |> map sk_cross_val_score |> p
```

sk_davies_bouldin

`sk_davies_bouldin` — batch 79 builtin.

```
my $result = sk_davies_bouldin $x
# or in a pipeline:
@list |> map sk_davies_bouldin |> p
```

sk_dbscan_fit

`sk_dbscan_fit` — batch 79 builtin.

```
my $result = sk_dbscan_fit $x
# or in a pipeline:
@list |> map sk_dbscan_fit |> p
```

sk_decision_tree_split

`sk_decision_tree_split` — batch 79 builtin.

```
my $result = sk_decision_tree_split $x
# or in a pipeline:
@list |> map sk_decision_tree_split |> p
```

sk_doc2vec_train

`sk_doc2vec_train` — batch 79 builtin.

```
my $result = sk_doc2vec_train $x
# or in a pipeline:
@list |> map sk_doc2vec_train |> p
```

sk_gbt_fit

`sk_gbt_fit` — batch 79 builtin.

```
my $result = sk_gbt_fit $x
# or in a pipeline:
@list |> map sk_gbt_fit |> p
```

sk_grid_search

`sk_grid_search` — batch 79 builtin.

```
my $result = sk_grid_search $x
# or in a pipeline:
@list |> map sk_grid_search |> p
```

sk_isolation_forest_fit

`sk_isolation_forest_fit` — batch 79 builtin.

```
my $result = sk_isolation_forest_fit $x
# or in a pipeline:
@list |> map sk_isolation_forest_fit |> p
```

sk_kfold_split

`sk_kfold_split` — batch 79 builtin.

```
my $result = sk_kfold_split $x
# or in a pipeline:
@list |> map sk_kfold_split |> p
```

sk_kmeans_fit

`sk_kmeans_fit` — batch 79 builtin.

```
my $result = sk_kmeans_fit $x
# or in a pipeline:
@list |> map sk_kmeans_fit |> p
```

sk_knn_predict

`sk_knn_predict` — batch 79 builtin.

```
my $result = sk_knn_predict $x
# or in a pipeline:
@list |> map sk_knn_predict |> p
```

sk_label_encode

`sk_label_encode` — batch 79 builtin.

```
my $result = sk_label_encode $x
# or in a pipeline:
@list |> map sk_label_encode |> p
```

sk_lda_topic

`sk_lda_topic` — batch 79 builtin.

```
my $result = sk_lda_topic $x
# or in a pipeline:
@list |> map sk_lda_topic |> p
```

sk_lightgbm_fit

`sk_lightgbm_fit` — batch 79 builtin.

```
my $result = sk_lightgbm_fit $x
# or in a pipeline:
@list |> map sk_lightgbm_fit |> p
```

sk_lof_fit

`sk_lof_fit` — batch 79 builtin.

```
my $result = sk_lof_fit $x
# or in a pipeline:
@list |> map sk_lof_fit |> p
```

sk_logistic_fit

`sk_logistic_fit` — batch 79 builtin.

```
my $result = sk_logistic_fit $x
# or in a pipeline:
@list |> map sk_logistic_fit |> p
```

sk_logistic_predict

`sk_logistic_predict` — batch 79 builtin.

```
my $result = sk_logistic_predict $x
# or in a pipeline:
@list |> map sk_logistic_predict |> p
```

sk_min_max_scaler

`sk_min_max_scaler` — batch 79 builtin.

```
my $result = sk_min_max_scaler $x
# or in a pipeline:
@list |> map sk_min_max_scaler |> p
```

sk_mutual_info

`sk_mutual_info` — batch 79 builtin.

```
my $result = sk_mutual_info $x
# or in a pipeline:
@list |> map sk_mutual_info |> p
```

sk_naive_bayes_predict

`sk_naive_bayes_predict` — batch 79 builtin.

```
my $result = sk_naive_bayes_predict $x
# or in a pipeline:
@list |> map sk_naive_bayes_predict |> p
```

sk_nmf_topic

`sk_nmf_topic` — batch 79 builtin.

```
my $result = sk_nmf_topic $x
# or in a pipeline:
@list |> map sk_nmf_topic |> p
```

sk_one_hot

sk_one_hot — batch 79 builtin.

```
my $result = sk_one_hot $x
# or in a pipeline:
@list |> map sk_one_hot |> p
```

sk_ordinal_encode

sk_ordinal_encode — batch 79 builtin.

```
my $result = sk_ordinal_encode $x
# or in a pipeline:
@list |> map sk_ordinal_encode |> p
```

sk_pca_fit

sk_pca_fit — batch 79 builtin.

```
my $result = sk_pca_fit $x
# or in a pipeline:
@list |> map sk_pca_fit |> p
```

sk_pipeline_fit

sk_pipeline_fit — batch 79 builtin.

```
my $result = sk_pipeline_fit $x
# or in a pipeline:
@list |> map sk_pipeline_fit |> p
```

sk_power_transform

sk_power_transform — batch 79 builtin.

```
my $result = sk_power_transform $x
# or in a pipeline:
@list |> map sk_power_transform |> p
```

sk_quantile_transform

sk_quantile_transform — batch 79 builtin.

```
my $result = sk_quantile_transform $x
# or in a pipeline:
@list |> map sk_quantile_transform |> p
```

sk_random_forest_fit

sk_random_forest_fit — batch 79 builtin.

```
my $result = sk_random_forest_fit $x
# or in a pipeline:
@list |> map sk_random_forest_fit |> p
```

sk_random_search

sk_random_search — batch 79 builtin.

```
my $result = sk_random_search $x
# or in a pipeline:
@list |> map sk_random_search |> p
```

sk_robust_scaler

`sk_robust_scaler` — batch 79 builtin.

```
my $result = sk_robust_scaler $x
# or in a pipeline:
@list |> map sk_robust_scaler |> p
```

sk_silhouette

`sk_silhouette` — batch 79 builtin.

```
my $result = sk_silhouette $x
# or in a pipeline:
@list |> map sk_silhouette |> p
```

sk_standard_scaler

`sk_standard_scaler` — batch 79 builtin.

```
my $result = sk_standard_scaler $x
# or in a pipeline:
@list |> map sk_standard_scaler |> p
```

sk_stratified_kfold

`sk_stratified_kfold` — batch 79 builtin.

```
my $result = sk_stratified_kfold $x
# or in a pipeline:
@list |> map sk_stratified_kfold |> p
```

sk_svm_fit

`sk_svm_fit` — batch 79 builtin.

```
my $result = sk_svm_fit $x
# or in a pipeline:
@list |> map sk_svm_fit |> p
```

sk_tfidf

`sk_tfidf` — batch 79 builtin.

```
my $result = sk_tfidf $x
# or in a pipeline:
@list |> map sk_tfidf |> p
```

sk_tsne_fit

`sk_tsne_fit` — batch 79 builtin.

```
my $result = sk_tsne_fit $x
# or in a pipeline:
@list |> map sk_tsne_fit |> p
```

sk_umap_fit

`sk_umap_fit` — batch 79 builtin.

```
my $result = sk_umap_fit $x
# or in a pipeline:
@list |> map sk_umap_fit |> p
```

sk_word2vec_train

`sk_word2vec_train` — batch 79 builtin.

```
my $result = sk_word2vec_train $x
# or in a pipeline:
@list |> map sk_word2vec_train |> p
```

sk_xgb_fit

`sk_xgb_fit` — batch 79 builtin.

```
my $result = sk_xgb_fit $x
# or in a pipeline:
@list |> map sk_xgb_fit |> p
```

skew

`skew` — math / numeric (uncategorized batch) builtin. Alias for `skewness`.

```
my $result = skew $x
# or in a pipeline:
@list |> map skew |> p
```

skewness

`skewness` — math / numeric (uncategorized batch) builtin.

```
my $result = skewness $x
# or in a pipeline:
@list |> map skewness |> p
```

skill_score

`skill_score` — batch 57 builtin.

```
my $result = skill_score $x
# or in a pipeline:
@list |> map skill_score |> p
```

skin_depth

`skin_depth` — batch 12 builtin.

```
my $result = skin_depth $x
# or in a pipeline:
@list |> map skin_depth |> p
```

skip_assert

`skip_assert` — testing framework builtin. Alias for `test_skip`.

```
my $result = skip_assert $x
# or in a pipeline:
@list |> map skip_assert |> p
```

skip_test

`skip_test` — testing framework builtin. Alias for `test_skip`.

```
my $result = skip_test $x
# or in a pipeline:
@list |> map skip_test |> p
```

skip_while

`skip_while { COND } LIST` — skip leading elements while the predicate is true (alias for `drop_while`).

Behavior is identical to `drop_while`: once the predicate returns false, that element and all subsequent elements are emitted. The `skip_while` name is provided for users coming from Rust or Kotlin where this is the conventional name. Both compile to the same streaming operation internally.

```
1:10 |> skip_while { _ < 5 } |> e p # 5 6 7 8 9 10
@sorted |> skip_while { _ le "m" } |> e p
```

sla

`sla` — python/ruby stdlib builtin. Alias for `slice_arr`.

```
my $result = sla $x
# or in a pipeline:
@list |> map sla |> p
```

slca

`slca` — additional missing stdlib functions builtin. Alias for `slice_after`.

```
my $result = slca $x
# or in a pipeline:
@list |> map slca |> p
```

slcb

`slcb` — additional missing stdlib functions builtin. Alias for `slice_before`.

```
my $result = slcb $x
# or in a pipeline:
@list |> map slcb |> p
```

slcw

`slcw` — additional missing stdlib functions builtin. Alias for `slice_when`.

```
my $result = slcw $x
# or in a pipeline:
@list |> map slcw |> p
```

slic_superpixels

`slic_superpixels` — batch 77 builtin.

```
my $result = slic_superpixels $x
# or in a pipeline:
@list |> map slic_superpixels |> p
```

slice_after

`slice_after` — additional missing stdlib functions builtin.

```
my $result = slice_after $x
# or in a pipeline:
@list |> map slice_after |> p
```

slice_arr

`slice_arr` — python/ruby stdlib builtin.

```
my $result = slice_arr $x
# or in a pipeline:
@list |> map slice_arr |> p
```

slice_before

`slice_before` — additional missing stdlib functions builtin.

```
my $result = slice_before $x
# or in a pipeline:
@list |> map slice_before |> p
```

slice_when

`slice_when` — additional missing stdlib functions builtin.

```
my $result = slice_when $x
# or in a pipeline:
@list |> map slice_when |> p
```

sliced_wasserstein

`sliced_wasserstein` — batch 40 builtin.

```
my $result = sliced_wasserstein $x
# or in a pipeline:
@list |> map sliced_wasserstein |> p
```

sliding_pairs

`sliding_pairs` — collection more builtin.

```
my $result = sliding_pairs $x
# or in a pipeline:
@list |> map sliding_pairs |> p
```

sliding_window

`sliding_window` — functional / iterator builtin.

```
my $result = sliding_window $x
# or in a pipeline:
@list |> map sliding_window |> p
```

slope

`slope` — geometry / physics builtin.

```
my $result = slope $x
# or in a pipeline:
@list |> map slope |> p
```

slug

`slug` — extended stdlib builtin. Alias for `slugify`.

```
my $result = slug $x
# or in a pipeline:
@list |> map slug |> p
```

slugify

`slugify` — extended stdlib builtin.

```
my $result = slugify $x
# or in a pipeline:
@list |> map slugify |> p
```

slutsky_decompose

`slutsky_decompose` — batch 73 builtin.

```
my $result = slutsky_decompose $x
# or in a pipeline:
@list |> map slutsky_decompose |> p
```

smap

`smap` — python/ruby stdlib builtin. Alias for `starmap`.

```
my $result = smap $x
# or in a pipeline:
@list |> map smap |> p
```

smembers

`smembers` — batch 71 builtin.

```
my $result = smembers $x
# or in a pipeline:
@list |> map smembers |> p
```

smith_normal_2x2_step

`smith_normal_2x2_step` — batch 35 builtin.

```
my $result = smith_normal_2x2_step $x
# or in a pipeline:
@list |> map smith_normal_2x2_step |> p
```

smith_predictor_step

`smith_predictor_step` — batch 59 builtin.

```
my $result = smith_predictor_step $x
# or in a pipeline:
@list |> map smith_predictor_step |> p
```

smith_q

`smith_q` — batch 11 builtin.

```
my $result = smith_q $x
# or in a pipeline:
@list |> map smith_q |> p
```

smith_waterman_score

`smith_waterman_score` — batch 33 builtin.

```
my $result = smith_waterman_score $x
# or in a pipeline:
@list |> map smith_waterman_score |> p
```

smmap

`smmap` — stress / telemetry extensions builtin. Alias for `stress_mmap`.

```
my $result = smmap $x
# or in a pipeline:
@list |> map smmap |> p
```

smoothstep

`smoothstep($edge0, $edge1, $x)` — smooth Hermite interpolation. Returns 0 if x ⩽ edge0, 1 if x ⩾ edge1, otherwise smooth S-curve. Great for animations.

```
p smoothstep(0, 1, 0.5)   # 0.5 (but with smooth acceleration/deceleration)
p smoothstep(10, 20, 15)  # 0.5
```

smst

`smst` — extended stdlib builtin. Alias for `smoothstep`.

```
my $result = smst $x
# or in a pipeline:
@list |> map smst |> p
```

smt_qf_lia_solve_step

`smt_qf_lia_solve_step` — batch 67 builtin.

```
my $result = smt_qf_lia_solve_step $x
# or in a pipeline:
@list |> map smt_qf_lia_solve_step |> p
```

smt_qf_uf_combine

`smt_qf_uf_combine` — batch 67 builtin.

```
my $result = smt_qf_uf_combine $x
# or in a pipeline:
@list |> map smt_qf_uf_combine |> p
```

snake_to_camel

`snake_to_camel` — string processing (uncategorized batch) builtin.

```
my $result = snake_to_camel $input
```

snappy_encode

`snappy_encode` — batch 81 builtin.

```
my $result = snappy_encode $x
# or in a pipeline:
@list |> map snappy_encode |> p
```

snd

`snd` — algebraic match builtin. Alias for `second`.

```
my $result = snd $x
# or in a pipeline:
@list |> map snd |> p
```

snell

`snells_law($n1, $n2, $theta1)` (alias `snell`) — refraction angle using Snell's law. Angles in degrees. Returns undef for total internal reflection.

```
p snell(1, 1.5, 30)     # ~19.5° (air to glass)
p snell(1.5, 1, 45)     # undef (total internal reflection)
```

snell_theta2

`snell_theta2` — batch 23 builtin.

```
my $result = snell_theta2 $x
# or in a pipeline:
@list |> map snell_theta2 |> p
```

snowball_stem_english

`snowball_stem_english` — batch 69 builtin.

```
my $result = snowball_stem_english $x
# or in a pipeline:
@list |> map snowball_stem_english |> p
```

snowball_stem_french

`snowball_stem_french` — batch 69 builtin.

```
my $result = snowball_stem_french $x
# or in a pipeline:
@list |> map snowball_stem_french |> p
```

sobel_diagonal_kernel

`sobel_diagonal_kernel` — batches 14-16 builtin.

```
my $result = sobel_diagonal_kernel $x
# or in a pipeline:
@list |> map sobel_diagonal_kernel |> p
```

sobel_kernel_value

`sobel_kernel_value` — batch 40 builtin.

```
my $result = sobel_kernel_value $x
# or in a pipeline:
@list |> map sobel_kernel_value |> p
```

soc_estimate_coulomb

`soc_estimate_coulomb` — batch 38 builtin.

```
my $result = soc_estimate_coulomb $x
# or in a pipeline:
@list |> map soc_estimate_coulomb |> p
```

social_welfare_nash

`social_welfare_nash` — batch 73 builtin.

```
my $result = social_welfare_nash $x
# or in a pipeline:
@list |> map social_welfare_nash |> p
```

social_welfare_rawls

`social_welfare_rawls` — batch 73 builtin.

```
my $result = social_welfare_rawls $x
# or in a pipeline:
@list |> map social_welfare_rawls |> p
```

social_welfare_sum

`social_welfare_sum` — batch 43 builtin.

```
my $result = social_welfare_sum $x
# or in a pipeline:
@list |> map social_welfare_sum |> p
```

social_welfare_utilitarian

`social_welfare_utilitarian` — batch 73 builtin.

```
my $result = social_welfare_utilitarian $x
# or in a pipeline:
@list |> map social_welfare_utilitarian |> p
```

socketpair

`socketpair` — socket builtin.

```
my $result = socketpair $x
# or in a pipeline:
@list |> map socketpair |> p
```

sockets

`sockets` — filesystem extensions builtin.

```
my $result = sockets $x
# or in a pipeline:
@list |> map sockets |> p
```

sod

`sod` — extended stdlib builtin. Alias for `start_of_day`.

```
my $result = sod $x
# or in a pipeline:
@list |> map sod |> p
```

softmax

`softmax` — functional combinators builtin.

```
my $result = softmax $x
# or in a pipeline:
@list |> map softmax |> p
```

softplus

`softplus` applies the Softplus activation: ln(1 + e^x). Smooth approximation to ReLU.

```
p softplus(0)   # 0.6931 (ln 2)
p softplus(10)  # ≈ 10
```

softsign

`softsign` — batch 27 builtin.

```
my $result = softsign $x
# or in a pipeline:
@list |> map softsign |> p
```

soh

`soh` — extended stdlib builtin. Alias for `start_of_hour`.

```
my $result = soh $x
# or in a pipeline:
@list |> map soh |> p
```

soh_capacity_fade

`soh_capacity_fade` — batch 38 builtin.

```
my $result = soh_capacity_fade $x
# or in a pipeline:
@list |> map soh_capacity_fade |> p
```

soi_oscillation_index

`soi_oscillation_index` — batch 42 builtin.

```
my $result = soi_oscillation_index $x
# or in a pipeline:
@list |> map soi_oscillation_index |> p
```

sol

`sol` — constants builtin.

```
my $result = sol $x
# or in a pipeline:
@list |> map sol |> p
```

solar_constant

`solar_constant` — batches 14-16 builtin.

```
my $result = solar_constant $x
# or in a pipeline:
@list |> map solar_constant |> p
```

solar_constant_at_distance

`solar_constant_at_distance` — batch 42 builtin.

```
my $result = solar_constant_at_distance $x
# or in a pipeline:
@list |> map solar_constant_at_distance |> p
```

solar_cycle_phase

`solar_cycle_phase` — batch 42 builtin.

```
my $result = solar_cycle_phase $x
# or in a pipeline:
@list |> map solar_cycle_phase |> p
```

solar_declination

`solar_declination` — batch 9 builtin.

```
my $result = solar_declination $x
# or in a pipeline:
@list |> map solar_declination |> p
```

solar_luminosity_to_w

`solar_luminosity_to_w` — batch 30 builtin.

```
my $result = solar_luminosity_to_w $input
```

solar_mass_to_kg

`solar_mass_to_kg` — batch 30 builtin.

```
my $result = solar_mass_to_kg $input
```

solar_zenith_angle

`solar_zenith_angle` — batches 14-16 builtin.

```
my $result = solar_zenith_angle $x
# or in a pipeline:
@list |> map solar_zenith_angle |> p
```

sole

`sole` — ruby enumerable extras builtin.

```
my $result = sole $x
# or in a pipeline:
@list |> map sole |> p
```

solid_electrolyte_capacity

`solid_electrolyte_capacity` — batch 38 builtin.

```
my $result = solid_electrolyte_capacity $x
# or in a pipeline:
@list |> map solid_electrolyte_capacity |> p
```

solstice_winter

`solstice_winter` — batch 82 builtin.

```
my $result = solstice_winter $x
# or in a pipeline:
@list |> map solstice_winter |> p
```

solution_resistance_estimate

`solution_resistance_estimate` — batch 38 builtin.

```
my $result = solution_resistance_estimate $x
# or in a pipeline:
@list |> map solution_resistance_estimate |> p
```

solvable_length_upper

`solvable_length_upper` — batch 37 builtin.

```
my $result = solvable_length_upper $x
# or in a pipeline:
@list |> map solvable_length_upper |> p
```

solve_ivp_step

`solve_ivp_step` — batch 72 builtin.

```
my $result = solve_ivp_step $x
# or in a pipeline:
@list |> map solve_ivp_step |> p
```

som

`som` — extended stdlib builtin. Alias for `start_of_minute`.

```
my $result = som $x
# or in a pipeline:
@list |> map som |> p
```

some

`some` — algebraic match builtin.

```
my $result = some $x
# or in a pipeline:
@list |> map some |> p
```

sorensen_dice

`sorensen_dice` — batch 22 builtin.

```
my $result = sorensen_dice $x
# or in a pipeline:
@list |> map sorensen_dice |> p
```

sort_by

`sort_by` — functional / iterator builtin.

```
my $result = sort_by $x
# or in a pipeline:
@list |> map sort_by |> p
```

sort_on

`sort_on` — haskell list functions builtin.

```
my $result = sort_on $x
# or in a pipeline:
@list |> map sort_on |> p
```

sort_words

`sort_words` — string processing (uncategorized batch) builtin.

```
my $result = sort_words $x
# or in a pipeline:
@list |> map sort_words |> p
```

sorted

`sorted` — file stat / path builtin.

```
my $result = sorted $x
# or in a pipeline:
@list |> map sorted |> p
```

sorted_by_length

`sorted_by_length` — file stat / path builtin.

```
my $result = sorted_by_length $x
# or in a pipeline:
@list |> map sorted_by_length |> p
```

sorted_desc

`sorted_desc` — file stat / path builtin.

```
my $result = sorted_desc $x
# or in a pipeline:
@list |> map sorted_desc |> p
```

sorted_nums

`sorted_nums` — file stat / path builtin.

```
my $result = sorted_nums $x
# or in a pipeline:
@list |> map sorted_nums |> p
```

sortino

`sortino_ratio(\@returns, $target)` (alias `sortino`) — computes the Sortino ratio, similar to Sharpe but only penalizes downside volatility. Better for asymmetric return distributions.

```
my @returns = (0.05, -0.02, 0.08, -0.01)
p sortino(\@returns, 0.0)   # penalizes only negative returns
```

sortw

`sortw` — string processing (uncategorized batch) builtin. Alias for `sort_words`.

```
my $result = sortw $x
# or in a pipeline:
@list |> map sortw |> p
```

sos2tf_step

`sos2tf_step` — batch 74 builtin.

```
my $result = sos2tf_step $x
# or in a pipeline:
@list |> map sos2tf_step |> p
```

sos_constraint_check

`sos_constraint_check` — batch 41 builtin.

```
my $result = sos_constraint_check $x
# or in a pipeline:
@list |> map sos_constraint_check |> p
```

sosfilt_step

`sosfilt_step` — batch 74 builtin.

```
my $result = sosfilt_step $x
# or in a pipeline:
@list |> map sosfilt_step |> p
```

sound_db

`sound_db` — batch 23 builtin.

```
my $result = sound_db $x
# or in a pipeline:
@list |> map sound_db |> p
```

sound_gas

`sound_gas` — batch 23 builtin.

```
my $result = sound_gas $x
# or in a pipeline:
@list |> map sound_gas |> p
```

sound_horizon_recomb

`sound_horizon_recomb` — batch 31 builtin.

```
my $result = sound_horizon_recomb $x
# or in a pipeline:
@list |> map sound_horizon_recomb |> p
```

sound_solid

`sound_solid` — batch 23 builtin.

```
my $result = sound_solid $x
# or in a pipeline:
@list |> map sound_solid |> p
```

soundex

`soundex` — extended stdlib builtin.

```
my $result = soundex $x
# or in a pipeline:
@list |> map soundex |> p
```

soundex_phonetic

`soundex_phonetic` — batch 69 builtin.

```
my $result = soundex_phonetic $x
# or in a pipeline:
@list |> map soundex_phonetic |> p
```

spaceship

`spaceship` — file stat / path builtin.

```
my $result = spaceship $x
# or in a pipeline:
@list |> map spaceship |> p
```

span

`span` — list helpers (batch 4) builtin.

```
my $result = span $x
# or in a pipeline:
@list |> map span |> p
```

span_fn

`span_fn` — haskell list functions builtin.

```
my $result = span_fn $x
# or in a pipeline:
@list |> map span_fn |> p
```

spanf

`spanf` — haskell list functions builtin. Alias for `span_fn`.

```
my $result = spanf $x
# or in a pipeline:
@list |> map spanf |> p
```

sparse_csr_build

`sparse_csr_build` — batches 17-18 builtin.

```
my $result = sparse_csr_build $x
# or in a pipeline:
@list |> map sparse_csr_build |> p
```

sparse_csr_mul_vec

`sparse_csr_mul_vec` — batches 17-18 builtin.

```
my $result = sparse_csr_mul_vec $x
# or in a pipeline:
@list |> map sparse_csr_mul_vec |> p
```

sparse_density

`sparse_density` — batches 17-18 builtin.

```
my $result = sparse_density $x
# or in a pipeline:
@list |> map sparse_density |> p
```

spat

`spat` — algebraic match builtin. Alias for `split_at`.

```
my $result = spat $x
# or in a pipeline:
@list |> map spat |> p
```

spbfs

`spbfs` — extended stdlib builtin. Alias for `shortest_path_bfs`.

```
my $result = spbfs $x
# or in a pipeline:
@list |> map spbfs |> p
```

spearman

`spearman_correlation(\@x, \@y)` (alias `spearman`) — computes Spearman's rank correlation coefficient between two datasets. Measures monotonic relationships, robust to outliers. Returns a value from -1 to 1.

```
my @x = (1, 2, 3, 4, 5)
my @y = (5, 6, 7, 8, 7)
p spearman(\@x, \@y)   # high positive correlation
```

spearmanr

`spearmanr` — batch 53 builtin.

```
my $result = spearmanr $x
# or in a pipeline:
@list |> map spearmanr |> p
```

species_area

`species_area` — batch 22 builtin.

```
my $result = species_area $x
# or in a pipeline:
@list |> map species_area |> p
```

specific_angular_momentum

`specific_angular_momentum` — batches 14-16 builtin.

```
my $result = specific_angular_momentum $x
# or in a pipeline:
@list |> map specific_angular_momentum |> p
```

specific_capacity_active

`specific_capacity_active` — batch 38 builtin.

```
my $result = specific_capacity_active $x
# or in a pipeline:
@list |> map specific_capacity_active |> p
```

specific_gravity_water

`specific_gravity_water` — batch 65 builtin.

```
my $result = specific_gravity_water $x
# or in a pipeline:
@list |> map specific_gravity_water |> p
```

specific_heat_const_v

`specific_heat_const_v` — batch 12 builtin.

```
my $result = specific_heat_const_v $x
# or in a pipeline:
@list |> map specific_heat_const_v |> p
```

specific_orbital_energy

`specific_orbital_energy` — batches 14-16 builtin.

```
my $result = specific_orbital_energy $x
# or in a pipeline:
@list |> map specific_orbital_energy |> p
```

specificity

`specificity` — batch 27 builtin.

```
my $result = specificity $x
# or in a pipeline:
@list |> map specificity |> p
```

speck_round

`speck_round` — batches 14-16 builtin.

```
my $result = speck_round $x
# or in a pipeline:
@list |> map speck_round |> p
```

spectral_density_estimate

`spectral_density_estimate` — batch 78 builtin.

```
my $result = spectral_density_estimate $x
# or in a pipeline:
@list |> map spectral_density_estimate |> p
```

spectral_kappa_minus53

spectral_kappa_minus53 — batch 42 builtin.

```
my $result = spectral_kappa_minus53 $x
# or in a pipeline:
@list |> map spectral_kappa_minus53 |> p
```

spectral_radius

spectral_radius — batch 34 builtin.

```
my $result = spectral_radius $x
# or in a pipeline:
@list |> map spectral_radius |> p
```

spectral_signature_match

spectral_signature_match — batch 40 builtin.

```
my $result = spectral_signature_match $x
# or in a pipeline:
@list |> map spectral_signature_match |> p
```

spectro

spectro — dsp / signal (extended) builtin. Alias for spectrogram.

```
my $result = spectro $x
# or in a pipeline:
@list |> map spectro |> p
```

spectrogram

spectrogram(\@signal, $window_size, $hop_size) (alias stft) — computes Short-Time Fourier Transform. Returns a 2D arrayref where each row is the spectrum of a windowed segment. Useful for time-frequency analysis.

```
my $spec = stft(\@audio, 1024, 512)
# $spec->[t][f] is magnitude at time t, frequency f
```

spectrogram_simple

spectrogram_simple — batch 25 builtin.

```
my $result = spectrogram_simple $x
# or in a pipeline:
@list |> map spectrogram_simple |> p
```

spectrophotometer_beer_lambert

spectrophotometer_beer_lambert — batch 63 builtin.

```
my $result = spectrophotometer_beer_lambert $x
# or in a pipeline:
@list |> map spectrophotometer_beer_lambert |> p
```

spectroscopic_factor

spectroscopic_factor — batch 63 builtin.

```
my $result = spectroscopic_factor $x
# or in a pipeline:
@list |> map spectroscopic_factor |> p
```

speed_distance_time

`speed_distance_time` — physics formulas builtin.

```
my $result = speed_distance_time $x
# or in a pipeline:
@list |> map speed_distance_time |> p
```

speed_of_light

`speed_of_light` — math / physics constants builtin.

```
my $result = speed_of_light $x
# or in a pipeline:
@list |> map speed_of_light |> p
```

speed_of_sound_ideal

`speed_of_sound_ideal` — batch 12 builtin.

```
my $result = speed_of_sound_ideal $x
# or in a pipeline:
@list |> map speed_of_sound_ideal |> p
```

spew

`spew` — filesystem extensions builtin. Alias for `spurt`.

```
my $result = spew $x
# or in a pipeline:
@list |> map spew |> p
```

sph_harm

`sph_harm` — batch 72 builtin.

```
my $result = sph_harm $x
# or in a pipeline:
@list |> map sph_harm |> p
```

sphenic_q

`sphenic_q` — batch 10 builtin.

```
my $result = sphenic_q $x
# or in a pipeline:
@list |> map sphenic_q |> p
```

sphere_of_influence

`sphere_of_influence` — batch 31 builtin.

```
my $result = sphere_of_influence $x
# or in a pipeline:
@list |> map sphere_of_influence |> p
```

sphere_surface

sphere_surface — geometry / physics builtin.

```
my $result = sphere_surface $x
# or in a pipeline:
@list |> map sphere_surface |> p
```

sphere_volume

sphere_volume — geometry / physics builtin.

```
my $result = sphere_volume $x
# or in a pipeline:
@list |> map sphere_volume |> p
```

spherical_bessel_j0

spherical_bessel_j0 — batch 29 builtin.

```
my $result = spherical_bessel_j0 $x
# or in a pipeline:
@list |> map spherical_bessel_j0 |> p
```

spherical_bessel_j1

spherical_bessel_j1 — batch 29 builtin.

```
my $result = spherical_bessel_j1 $x
# or in a pipeline:
@list |> map spherical_bessel_j1 |> p
```

spherical_bessel_y0

spherical_bessel_y0 — batch 29 builtin.

```
my $result = spherical_bessel_y0 $x
# or in a pipeline:
@list |> map spherical_bessel_y0 |> p
```

spherical_bessel_y1

spherical_bessel_y1 — batch 29 builtin.

```
my $result = spherical_bessel_y1 $x
# or in a pipeline:
@list |> map spherical_bessel_y1 |> p
```

spherical_jn

spherical_jn — batch 72 builtin.

```
my $result = spherical_jn $x
# or in a pipeline:
@list |> map spherical_jn |> p
```

spherical_triangle_area

spherical_triangle_area — batch 28 builtin.

```
my $result = spherical_triangle_area $x
# or in a pipeline:
@list |> map spherical_triangle_area |> p
```

spherical_yn

`spherical_yn` — batch 72 builtin.

```
my $result = spherical_yn $x
# or in a pipeline:
@list |> map spherical_yn |> p
```

sphsurf

`sphsurf` — geometry / physics builtin. Alias for `sphere_surface`.

```
my $result = sphsurf $x
# or in a pipeline:
@list |> map sphsurf |> p
```

sphvol

`sphvol` — geometry / physics builtin. Alias for `sphere_volume`.

```
my $result = sphvol $x
# or in a pipeline:
@list |> map sphvol |> p
```

spin_casimir

`spin_casimir` — batch 32 builtin.

```
my $result = spin_casimir $x
# or in a pipeline:
@list |> map spin_casimir |> p
```

spin_orbit_coupling_term

`spin_orbit_coupling_term` — batch 39 builtin.

```
my $result = spin_orbit_coupling_term $x
# or in a pipeline:
@list |> map spin_orbit_coupling_term |> p
```

spin_spin_coupling_term

`spin_spin_coupling_term` — batch 39 builtin.

```
my $result = spin_spin_coupling_term $x
# or in a pipeline:
@list |> map spin_spin_coupling_term |> p
```

spinner_start

`spinner_start` — pipeline / string helpers builtin.

```
my $result = spinner_start $x
# or in a pipeline:
@list |> map spinner_start |> p
```

spinner_stop

`spinner_stop` — pipeline / string helpers builtin.

```
my $result = spinner_stop $x
# or in a pipeline:
@list |> map spinner_stop |> p
```

split_at

`split_at` — algebraic match builtin.

```
my $result = split_at $x
# or in a pipeline:
@list |> map split_at |> p
```

split_on

`split_on` — go/general functional utilities builtin.

```
my $result = split_on $x
# or in a pipeline:
@list |> map split_on |> p
```

split_regex

`split_regex` — extended stdlib builtin.

```
my $result = split_regex $x
# or in a pipeline:
@list |> map split_regex |> p
```

split_with

`split_with` — algebraic match builtin.

```
my $result = split_with $x
# or in a pipeline:
@list |> map split_with |> p
```

splitmix64_step

`splitmix64_step` — batch 26 builtin.

```
my $result = splitmix64_step $x
# or in a pipeline:
@list |> map splitmix64_step |> p
```

splre

`splre` — extended stdlib builtin. Alias for `split_regex`.

```
my $result = splre $x
# or in a pipeline:
@list |> map splre |> p
```

spon

`spon` — go/general functional utilities builtin. Alias for `split_on`.

```
my $result = spon $x
# or in a pipeline:
@list |> map spon |> p
```

spookyhash_128

spookyhash_128 — batch 81 builtin.

```
my $result = spookyhash_128 $x
# or in a pipeline:
@list |> map spookyhash_128 |> p
```

spop

spop — batch 71 builtin.

```
my $result = spop $x
# or in a pipeline:
@list |> map spop |> p
```

spring_energy

spring_energy — physics formulas builtin.

```
my $result = spring_energy $x
# or in a pipeline:
@list |> map spring_energy |> p
```

spring_force

spring_force — physics formulas builtin.

```
my $result = spring_force $x
# or in a pipeline:
@list |> map spring_force |> p
```

spring_period

spring_period — batch 12 builtin.

```
my $result = spring_period $x
# or in a pipeline:
@list |> map spring_period |> p
```

springpe

springpe — physics formulas builtin. Alias for spring_energy.

```
my $result = springpe $x
# or in a pipeline:
@list |> map springpe |> p
```

spt_n_correction

spt_n_correction — batch 65 builtin.

```
my $result = spt_n_correction $x
# or in a pipeline:
@list |> map spt_n_correction |> p
```

spw

`spw` — algebraic match builtin. Alias for `split_with`.

```
my $result = spw $x
# or in a pipeline:
@list |> map spw |> p
```

sql

`sql` — data / network builtin. Alias for `sqlite`.

```
my $result = sql $x
# or in a pipeline:
@list |> map sql |> p
```

sqr

`sqr` — trig / math (batch 2) builtin.

```
my $result = sqr $x
# or in a pipeline:
@list |> map sqr |> p
```

sqrt2

`sqrt2` — math / physics constants builtin.

```
my $result = sqrt2 $x
# or in a pipeline:
@list |> map sqrt2 |> p
```

sqrt_each

`sqrt_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = sqrt_each $x
# or in a pipeline:
@list |> map sqrt_each |> p
```

sqrt_swap_gate

`sqrt_swap_gate` — batches 14-16 builtin.

```
my $result = sqrt_swap_gate $x
# or in a pipeline:
@list |> map sqrt_swap_gate |> p
```

square_each

`square_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = square_each $x
# or in a pipeline:
@list |> map square_each |> p
```

square_pyramidal_number

`square_pyramidal_number` — batch 10 builtin.

```
my $result = square_pyramidal_number $x
# or in a pipeline:
@list |> map square_pyramidal_number |> p
```

square_root

`square_root` — math functions builtin.

```
my $result = square_root $x
# or in a pipeline:
@list |> map square_root |> p
```

squared_hinge

`squared_hinge` — batch 27 builtin.

```
my $result = squared_hinge $x
# or in a pipeline:
@list |> map squared_hinge |> p
```

squared_triangular

`squared_triangular` — batch 11 builtin.

```
my $result = squared_triangular $x
# or in a pipeline:
@list |> map squared_triangular |> p
```

squares_seq

`squares_seq` — sequences builtin.

```
my $result = squares_seq $x
# or in a pipeline:
@list |> map squares_seq |> p
```

squeezing_db

`squeezing_db` — batch 32 builtin.

```
my $result = squeezing_db $x
# or in a pipeline:
@list |> map squeezing_db |> p
```

squish

`squish` — trivial string ops builtin.

```
my $result = squish $x
# or in a pipeline:
@list |> map squish |> p
```

sregex

`sregex` — stress / telemetry extensions builtin. Alias for `stress_regex`.

```
my $result = sregex $x
# or in a pipeline:
@list |> map sregex |> p
```

srem

`srem` — batch 71 builtin.

```
my $result = srem $x
# or in a pipeline:
@list |> map srem |> p
```

srgb_to_xyz

`srgb_to_xyz` — batch 30 builtin.

```
my $result = srgb_to_xyz $input
```

srton

`srton` — haskell list functions builtin. Alias for `sort_on`.

```
my $result = srton $x
# or in a pipeline:
@list |> map srton |> p
```

ssa_phi_insert

`ssa_phi_insert` — batch 68 builtin.

```
my $result = ssa_phi_insert $x
# or in a pipeline:
@list |> map ssa_phi_insert |> p
```

ssn_check

`ssn_check` — batch 82 builtin.

```
my $result = ssn_check $x
# or in a pipeline:
@list |> map ssn_check |> p
```

ssort

`ssort` — stress / telemetry extensions builtin. Alias for `stress_sort`.

```
my $result = ssort $x
# or in a pipeline:
@list |> map ssort |> p
```

sst_anomaly_step

`sst_anomaly_step` — batch 42 builtin.

```
my $result = sst_anomaly_step $x
# or in a pipeline:
@list |> map sst_anomaly_step |> p
```

stable_matching_count

`stable_matching_count` — batch 73 builtin.

```
my $result = stable_matching_count $x
# or in a pipeline:
@list |> map stable_matching_count |> p
```

stack_new

`stack_new` — data structure helpers builtin.

```
my $result = stack_new $x
# or in a pipeline:
@list |> map stack_new |> p
```

stackelberg_eq

`stackelberg_eq` — batch 73 builtin.

```
my $result = stackelberg_eq $x
# or in a pipeline:
@list |> map stackelberg_eq |> p
```

stackelberg_step

`stackelberg_step` — batch 43 builtin.

```
my $result = stackelberg_step $x
# or in a pipeline:
@list |> map stackelberg_step |> p
```

stag_hunt_payoff

`stag_hunt_payoff` — batch 43 builtin.

```
my $result = stag_hunt_payoff $x
# or in a pipeline:
@list |> map stag_hunt_payoff |> p
```

standard_error

`standard_error` — math / numeric (uncategorized batch) builtin.

```
my $result = standard_error $x
# or in a pipeline:
@list |> map standard_error |> p
```

standardized_mortality_smr

`standardized_mortality_smr` — batch 61 builtin.

```
my $result = standardized_mortality_smr $x
# or in a pipeline:
@list |> map standardized_mortality_smr |> p
```

standing_wave_fundamental

`standing_wave_fundamental` — batch 23 builtin.

```
my $result = standing_wave_fundamental $x
# or in a pipeline:
@list |> map standing_wave_fundamental |> p
```

star_number

`star_number` — batch 11 builtin.

```
my $result = star_number $x
# or in a pipeline:
@list |> map star_number |> p
```

stark_shift_linear

`stark_shift_linear` — batch 32 builtin.

```
my $result = stark_shift_linear $x
# or in a pipeline:
@list |> map stark_shift_linear |> p
```

starmap

`starmap` — python/ruby stdlib builtin.

```
my $result = starmap $x
# or in a pipeline:
@list |> map starmap |> p
```

start_of_day

`start_of_day` — extended stdlib builtin.

```
my $result = start_of_day $x
# or in a pipeline:
@list |> map start_of_day |> p
```

start_of_hour

`start_of_hour` — extended stdlib builtin.

```
my $result = start_of_hour $x
# or in a pipeline:
@list |> map start_of_hour |> p
```

start_of_minute

`start_of_minute` — extended stdlib builtin.

```
my $result = start_of_minute $x
# or in a pipeline:
@list |> map start_of_minute |> p
```

starts_with

`starts_with` — trivial string ops builtin.

```
my $result = starts_with $x
# or in a pipeline:
@list |> map starts_with |> p
```

starts_with_any

`starts_with_any` — string (batch 2) builtin.

```
my $result = starts_with_any $x
# or in a pipeline:
@list |> map starts_with_any |> p
```

state_norm

`state_norm` — batch 32 builtin.

```
my $result = state_norm $x
# or in a pipeline:
@list |> map state_norm |> p
```

state_normalize

`state_normalize` — batch 32 builtin.

```
my $result = state_normalize $x
# or in a pipeline:
@list |> map state_normalize |> p
```

state_of_charge_kalman

`state_of_charge_kalman` — batch 38 builtin.

```
my $result = state_of_charge_kalman $x
# or in a pipeline:
@list |> map state_of_charge_kalman |> p
```

state_space_kalman

`state_space_kalman` — batch 78 builtin.

```
my $result = state_space_kalman $x
# or in a pipeline:
@list |> map state_space_kalman |> p
```

stats_describe

`stats_describe` — statistics (extended) builtin. Alias for `describe`.

```
my $result = stats_describe $x
# or in a pipeline:
@list |> map stats_describe |> p
```

stderr_stat

`stderr_stat` — math / numeric (uncategorized batch) builtin. Alias for `standard_error`.

```
my $result = stderr_stat $x
# or in a pipeline:
@list |> map stderr_stat |> p
```

stefan_boltzmann

`stefan_boltzmann` — physics formulas builtin.

```
my $result = stefan_boltzmann $x
# or in a pipeline:
@list |> map stefan_boltzmann |> p
```

stefan_boltzmann_constant

`stefan_boltzmann_constant` — physics constants builtin.

```
my $result = stefan_boltzmann_constant $x
# or in a pipeline:
@list |> map stefan_boltzmann_constant |> p
```

stefan_boltzmann_radiation

`stefan_boltzmann_radiation` — batch 42 builtin.

```
my $result = stefan_boltzmann_radiation $x
# or in a pipeline:
@list |> map stefan_boltzmann_radiation |> p
```

stefan_boltzmann_si

`stefan_boltzmann_si` — batch 31 builtin.

```
my $result = stefan_boltzmann_si $x
# or in a pipeline:
@list |> map stefan_boltzmann_si |> p
```

stefboltz

`stefboltz` — physics formulas builtin. Alias for `stefan_boltzmann`.

```
my $result = stefboltz $x
# or in a pipeline:
@list |> map stefboltz |> p
```

steffensen

`steffensen` — batch 19 builtin. Alias for `steffensen_root`.

```
my $result = steffensen $x
# or in a pipeline:
@list |> map steffensen |> p
```

steffensen_root

`steffensen_root` — batch 19 builtin.

```
my $result = steffensen_root $x
# or in a pipeline:
@list |> map steffensen_root |> p
```

stem_lancaster

`stem_lancaster` — batch 69 builtin.

```
my $result = stem_lancaster $x
# or in a pipeline:
@list |> map stem_lancaster |> p
```

step

`step` — python/ruby stdlib builtin.

```
my $result = step $x
# or in a pipeline:
@list |> map step |> p
```

stern_layer_capacitance

`stern_layer_capacitance` — batch 38 builtin.

```
my $result = stern_layer_capacitance $x
# or in a pipeline:
@list |> map stern_layer_capacitance |> p
```

stft_step

`stft_step` — batch 74 builtin.

```
my $result = stft_step $x
# or in a pipeline:
@list |> map stft_step |> p
```

sthread

`sthread` — stress / telemetry extensions builtin. Alias for `stress_thread`.

```
my $result = sthread $x
# or in a pipeline:
@list |> map sthread |> p
```

stiffness_ratio

`stiffness_ratio` — batch 34 builtin.

```
my $result = stiffness_ratio $x
# or in a pipeline:
@list |> map stiffness_ratio |> p
```

stirling

`stirling` — math formulas builtin. Alias for `stirling_approx`.

```
my $result = stirling $x
# or in a pipeline:
@list |> map stirling |> p
```

stirling2

`stirling2` (alias `stirling_second`) computes the Stirling number of the second kind S(n,k) — the number of ways to partition n elements into k non-empty subsets.

```
p stirling2(4, 2)   # 7
p stirling2(5, 3)   # 25
```

stirling_approx

`stirling_approx` — math formulas builtin.

```
my $result = stirling_approx $x
# or in a pipeline:
@list |> map stirling_approx |> p
```

stl_decompose

`stl_decompose` — batch 78 builtin.

```
my $result = stl_decompose $x
# or in a pipeline:
@list |> map stl_decompose |> p
```

stoer_wagner_step

`stoer_wagner_step` — batch 75 builtin.

```
my $result = stoer_wagner_step $x
# or in a pipeline:
@list |> map stoer_wagner_step |> p
```

stone_to_kg

`stone_to_kg` — unit conversions builtin.

```
my $result = stone_to_kg $input
```

stormer_verlet_step

`stormer_verlet_step` — batch 34 builtin.

```
my $result = stormer_verlet_step $x
# or in a pipeline:
@list |> map stormer_verlet_step |> p
```

strang_split

`strang_split` — batch 34 builtin.

```
my $result = strang_split $x
# or in a pipeline:
@list |> map strang_split |> p
```

stratonovich_correction

`stratonovich_correction` — batch 34 builtin.

```
my $result = stratonovich_correction $x
# or in a pipeline:
@list |> map stratonovich_correction |> p
```

strdist

`strdist` — string processing (uncategorized batch) builtin. Alias for `string_distance`.

```
my $result = strdist $x
# or in a pipeline:
@list |> map strdist |> p
```

streak_color_index

`streak_color_index` — batch 65 builtin.

```
my $result = streak_color_index $x
# or in a pipeline:
@list |> map streak_color_index |> p
```

stream_prompt_cb

`stream_prompt_cb` — ai primitives (docs/ai_primitives.md) builtin.

```
my $result = stream_prompt_cb $x
# or in a pipeline:
@list |> map stream_prompt_cb |> p
```

streamfunction_step

`streamfunction_step` — batch 42 builtin.

```
my $result = streamfunction_step $x
# or in a pipeline:
@list |> map streamfunction_step |> p
```

strehl_ratio

`strehl_ratio` — batch 55 builtin.

```
my $result = strehl_ratio $x
# or in a pipeline:
@list |> map strehl_ratio |> p
```

strict_word_similarity

`strict_word_similarity` — batch 70 builtin.

```
my $result = strict_word_similarity $x
# or in a pipeline:
@list |> map strict_word_similarity |> p
```

strikethrough

`strikethrough` — color / ansi builtin. Alias for `red`.

```
my $result = strikethrough $x
# or in a pipeline:
@list |> map strikethrough |> p
```

string_agg

`string_agg` — batch 70 builtin.

```
my $result = string_agg $x
# or in a pipeline:
@list |> map string_agg |> p
```

string_camel_to_snake

`string_camel_to_snake` — batch 13 builtin.

```
my $result = string_camel_to_snake $input
```

string_chars

`string_chars` — batch 11 builtin.

```
my $result = string_chars $x
# or in a pipeline:
@list |> map string_chars |> p
```

string_count

`string_count` — string helpers (batch 4) builtin.

```
my $result = string_count $x
# or in a pipeline:
@list |> map string_count |> p
```

string_distance

`string_distance` — string processing (uncategorized batch) builtin.

```
my $result = string_distance $x
# or in a pipeline:
@list |> map string_distance |> p
```

string_drop_first

`string_drop_first` — batch 9 builtin.

```
my $result = string_drop_first $x
# or in a pipeline:
@list |> map string_drop_first |> p
```

string_drop_last

`string_drop_last` — batch 9 builtin.

```
my $result = string_drop_last $x
# or in a pipeline:
@list |> map string_drop_last |> p
```

string_drop_while

`string_drop_while` — batch 11 builtin.

```
my $result = string_drop_while $x
# or in a pipeline:
@list |> map string_drop_while |> p
```

string_expand_tabs

`string_expand_tabs` — batch 13 builtin.

```
my $result = string_expand_tabs $x
# or in a pipeline:
@list |> map string_expand_tabs |> p
```

string_intersperse

`string_intersperse` — batch 11 builtin.

```
my $result = string_intersperse $x
# or in a pipeline:
@list |> map string_intersperse |> p
```

string_kebab_to_snake

`string_kebab_to_snake` — batch 13 builtin.

```
my $result = string_kebab_to_snake $input
```

string_letter_frequency

`string_letter_frequency` — batch 11 builtin.

```
my $result = string_letter_frequency $x
# or in a pipeline:
@list |> map string_letter_frequency |> p
```

string_lines_count

`string_lines_count` — batch 11 builtin.

```
my $result = string_lines_count $x
# or in a pipeline:
@list |> map string_lines_count |> p
```

string_multiply

`string_multiply` — string processing (uncategorized batch) builtin.

```
my $result = string_multiply $x
# or in a pipeline:
@list |> map string_multiply |> p
```

string_normalize_spaces

`string_normalize_spaces` — batch 13 builtin.

```
my $result = string_normalize_spaces $x
# or in a pipeline:
@list |> map string_normalize_spaces |> p
```

string_partition_at_word

`string_partition_at_word` — batch 11 builtin.

```
my $result = string_partition_at_word $x
# or in a pipeline:
@list |> map string_partition_at_word |> p
```

string_replicate

`string_replicate` — batch 11 builtin.

```
my $result = string_replicate $x
# or in a pipeline:
@list |> map string_replicate |> p
```

string_snake_to_camel

`string_snake_to_camel` — batch 13 builtin.

```
my $result = string_snake_to_camel $input
```

string_snake_to_kebab

`string_snake_to_kebab` — batch 13 builtin.

```
my $result = string_snake_to_kebab $input
```

string_sort

`string_sort` — string helpers (batch 4) builtin.

```
my $result = string_sort $x
# or in a pipeline:
@list |> map string_sort |> p
```

string_split_at_first

`string_split_at_first` — batch 11 builtin.

```
my $result = string_split_at_first $x
# or in a pipeline:
@list |> map string_split_at_first |> p
```

string_take_first

`string_take_first` — batch 9 builtin.

```
my $result = string_take_first $x
# or in a pipeline:
@list |> map string_take_first |> p
```

string_take_last

`string_take_last` — batch 9 builtin.

```
my $result = string_take_last $x
# or in a pipeline:
@list |> map string_take_last |> p
```

string_take_while

`string_take_while` — batch 11 builtin.

```
my $result = string_take_while $x
# or in a pipeline:
@list |> map string_take_while |> p
```

string_to_charcodes

`string_to_charcodes` — batch 13 builtin.

```
my $result = string_to_charcodes $input
```

string_to_chars

`string_to_chars` — string helpers (batch 4) builtin.

```
my $result = string_to_chars $input
```

string_truncate_ellipsis

`string_truncate_ellipsis` — batch 13 builtin.

```
my $result = string_truncate_ellipsis $x
# or in a pipeline:
@list |> map string_truncate_ellipsis |> p
```

string_uniq_chars

`string_uniq_chars` — batch 11 builtin.

```
my $result = string_uniq_chars $x
# or in a pipeline:
@list |> map string_uniq_chars |> p
```

string_unique_chars

`string_unique_chars` — string helpers (batch 4) builtin.

```
my $result = string_unique_chars $x
# or in a pipeline:
@list |> map string_unique_chars |> p
```

string_words_count

`string_words_count` — batch 11 builtin.

```
my $result = string_words_count $x
# or in a pipeline:
@list |> map string_words_count |> p
```

string_xor

`string_xor` — batch 13 builtin.

```
my $result = string_xor $x
# or in a pipeline:
@list |> map string_xor |> p
```

strip_ansi

`strip_ansi` — color / ansi builtin.

```
my $result = strip_ansi $x
# or in a pipeline:
@list |> map strip_ansi |> p
```

strip_html

`strip_html` — string processing (uncategorized batch) builtin.

```
my $result = strip_html $x
# or in a pipeline:
@list |> map strip_html |> p
```

strip_prefix

`strip_prefix` — path helpers builtin.

```
my $result = strip_prefix $x
# or in a pipeline:
@list |> map strip_prefix |> p
```

strip_suffix

`strip_suffix` — path helpers builtin.

```
my $result = strip_suffix $x
# or in a pipeline:
@list |> map strip_suffix |> p
```

strlen

`strlen` — batch 71 builtin.

```
my $result = strlen $x
# or in a pipeline:
@list |> map strlen |> p
```

strmul

`strmul` — string processing (uncategorized batch) builtin. Alias for `string_multiply`.

```
my $result = strmul $x
# or in a pipeline:
@list |> map strmul |> p
```

strong_pseudoprime

`strong_pseudoprime` — batch 35 builtin.

```
my $result = strong_pseudoprime $x
# or in a pipeline:
@list |> map strong_pseudoprime |> p
```

strouhal_full

`strouhal_full` — batch 42 builtin.

```
my $result = strouhal_full $x
# or in a pipeline:
@list |> map strouhal_full |> p
```

strsm

`strsm` — batch 66 builtin.

```
my $result = strsm $x
# or in a pipeline:
@list |> map strsm |> p
```

struve_h0

`struve_h0` — batch 29 builtin.

```
my $result = struve_h0 $x
# or in a pipeline:
@list |> map struve_h0 |> p
```

struve_h1

`struve_h1` — batch 29 builtin.

```
my $result = struve_h1 $x
# or in a pipeline:
@list |> map struve_h1 |> p
```

student_pdf

t_pdf (alias tpdf) evaluates Student's t-distribution PDF at x with nu degrees of freedom.

```
p tpdf(0, 10)    # peak of t(10)
p tpdf(2.228, 10)
```

stulz_min_call

stulz_min_call — batch 20 builtin.

```
my $result = stulz_min_call $x
# or in a pipeline:
@list |> map stulz_min_call |> p
```

sturmian_word

sturmian_word — batches 14-16 builtin.

```
my $result = sturmian_word $x
# or in a pipeline:
@list |> map sturmian_word |> p
```

sub

The sub keyword is not supported in stryke. Use fn instead to define functions.

sub_script

sub_script — encoding / phonetics builtin. Alias for subscript.

```
my $result = sub_script $x
# or in a pipeline:
@list |> map sub_script |> p
```

subfact

subfact — math / numeric (uncategorized batch) builtin. Alias for subfactorial.

```
my $result = subfact $x
# or in a pipeline:
@list |> map subfact |> p
```

subfactorial

subfactorial — math / numeric (uncategorized batch) builtin.

```
my $result = subfactorial $x
# or in a pipeline:
@list |> map subfactorial |> p
```

subgame_perfect_eq

subgame_perfect_eq — batch 43 builtin.

```
my $result = subgame_perfect_eq $x
# or in a pipeline:
@list |> map subgame_perfect_eq |> p
```

subgradient_step_size

`subgradient_step_size` — batch 41 builtin.

```
my $result = subgradient_step_size $x
# or in a pipeline:
@list |> map subgradient_step_size |> p
```

subgraph_isomorphism

`subgraph_isomorphism` — batch 75 builtin.

```
my $result = subgraph_isomorphism $x
# or in a pipeline:
@list |> map subgraph_isomorphism |> p
```

submodular_curvature_bound

`submodular_curvature_bound` — batch 41 builtin.

```
my $result = submodular_curvature_bound $x
# or in a pipeline:
@list |> map submodular_curvature_bound |> p
```

submodular_greedy_step

`submodular_greedy_step` — batch 41 builtin.

```
my $result = submodular_greedy_step $x
# or in a pipeline:
@list |> map submodular_greedy_step |> p
```

subname

`subname` — list / aggregate builtin.

```
my $result = subname $x
# or in a pipeline:
@list |> map subname |> p
```

subscript

`subscript` — encoding / phonetics builtin.

```
my $result = subscript $x
# or in a pipeline:
@list |> map subscript |> p
```

subseqs

`subseqs` — additional missing stdlib functions builtin. Alias for `subsequences`.

```
my $result = subseqs $x
# or in a pipeline:
@list |> map subseqs |> p
```

subsequences

`subsequences` — additional missing stdlib functions builtin.

```
my $result = subsequences $x
# or in a pipeline:
@list |> map subsequences |> p
```

subset_construction

`subset_construction` — batch 68 builtin.

```
my $result = subset_construction $x
# or in a pipeline:
@list |> map subset_construction |> p
```

substitution_encrypt

`substitution_encrypt` — batch 26 builtin.

```
my $result = substitution_encrypt $x
# or in a pipeline:
@list |> map substitution_encrypt |> p
```

substring

`substring` — string helpers (batch 4) builtin.

```
my $result = substring $x
# or in a pipeline:
@list |> map substring |> p
```

substring_count

`substring_count` — batch 13 builtin.

```
my $result = substring_count $x
# or in a pipeline:
@list |> map substring_count |> p
```

substring_similarity

`substring_similarity` — batch 70 builtin.

```
my $result = substring_similarity $x
# or in a pipeline:
@list |> map substring_similarity |> p
```

subsumption_check

`subsumption_check` — batch 67 builtin.

```
my $result = subsumption_check $x
# or in a pipeline:
@list |> map subsumption_check |> p
```

succ

`succ` — conversion / utility (batch 4) builtin.

```
my $result = succ $x
# or in a pipeline:
@list |> map succ |> p
```

suffix_array_naive

`suffix_array_naive` — batch 10 builtin.

```
my $result = suffix_array_naive $x
# or in a pipeline:
@list |> map suffix_array_naive |> p
```

suffix_sums

`suffix_sums` — list helpers (batch 4) builtin.

```
my $result = suffix_sums $x
# or in a pipeline:
@list |> map suffix_sums |> p
```

sum_by

`sum_by` — python/ruby stdlib builtin.

```
my $result = sum_by $x
# or in a pipeline:
@list |> map sum_by |> p
```

sum_degrees

`sum_degrees` — batch 24 builtin.

```
my $result = sum_degrees $x
# or in a pipeline:
@list |> map sum_degrees |> p
```

sum_divisors

`sum_divisors` — math / numeric (uncategorized batch) builtin.

```
my $result = sum_divisors $x
# or in a pipeline:
@list |> map sum_divisors |> p
```

sum_list

`sum_list` — list helpers (batch 4) builtin.

```
my $result = sum_list $x
# or in a pipeline:
@list |> map sum_list |> p
```

sum_of

`sum_of` — more list helpers builtin.

```
my $result = sum_of $x
# or in a pipeline:
@list |> map sum_of |> p
```

sum_squares

`sum_squares` — extended stdlib builtin.

```
my $result = sum_squares $x
# or in a pipeline:
@list |> map sum_squares |> p
```

sum_two_squares

sum_two_squares — batch 35 builtin.

```
my $result = sum_two_squares $x
# or in a pipeline:
@list |> map sum_two_squares |> p
```

sumb

sumb — python/ruby stdlib builtin. Alias for sum_by.

```
my $result = sumb $x
# or in a pipeline:
@list |> map sumb |> p
```

sumif

sumif — batch 57 builtin.

```
my $result = sumif $x
# or in a pipeline:
@list |> map sumif |> p
```

sumifs

sumifs — batch 57 builtin.

```
my $result = sumifs $x
# or in a pipeline:
@list |> map sumifs |> p
```

sumproduct

sumproduct — batch 57 builtin.

```
my $result = sumproduct $x
# or in a pipeline:
@list |> map sumproduct |> p
```

sumsq

sumsq — extended stdlib builtin. Alias for sum_squares.

```
my $result = sumsq $x
# or in a pipeline:
@list |> map sumsq |> p
```

sun_distance_au

sun_distance_au — batch 55 builtin.

```
my $result = sun_distance_au $x
# or in a pipeline:
@list |> map sun_distance_au |> p
```

sun_position_low

`sun_position_low` — batch 55 builtin.

```
my $result = sun_position_low $x
# or in a pipeline:
@list |> map sun_position_low |> p
```

sun_radius

`sun_radius` — physics constants builtin.

```
my $result = sun_radius $x
# or in a pipeline:
@list |> map sun_radius |> p
```

sunion

`sunion` — batch 71 builtin.

```
my $result = sunion $x
# or in a pipeline:
@list |> map sunion |> p
```

sunspot_relative_number

`sunspot_relative_number` — batch 42 builtin.

```
my $result = sunspot_relative_number $x
# or in a pipeline:
@list |> map sunspot_relative_number |> p
```

sup

`sup` — encoding / phonetics builtin. Alias for `superscript`.

```
my $result = sup $x
# or in a pipeline:
@list |> map sup |> p
```

superbee_limiter

`superbee_limiter` — batch 34 builtin.

```
my $result = superbee_limiter $x
# or in a pipeline:
@list |> map superbee_limiter |> p
```

superdense_coding

`superdense_coding` — batch 80 builtin.

```
my $result = superdense_coding $x
# or in a pipeline:
@list |> map superdense_coding |> p
```

supergolden

`supergolden` — math constants builtin. Alias for `supergolden_ratio`.

```
my $result = supergolden $x
# or in a pipeline:
@list |> map supergolden |> p
```

supergolden_ratio

`supergolden_ratio` — math constants builtin.

```
my $result = supergolden_ratio $x
# or in a pipeline:
@list |> map supergolden_ratio |> p
```

superscript

`superscript` — encoding / phonetics builtin.

```
my $result = superscript $x
# or in a pipeline:
@list |> map superscript |> p
```

supply_elasticity

`supply_elasticity` — batch 73 builtin.

```
my $result = supply_elasticity $x
# or in a pipeline:
@list |> map supply_elasticity |> p
```

surf_keypoints

`surf_keypoints` — batch 77 builtin.

```
my $result = surf_keypoints $x
# or in a pipeline:
@list |> map surf_keypoints |> p
```

surface_wave_ms

`surface_wave_ms` — batch 65 builtin.

```
my $result = surface_wave_ms $x
# or in a pipeline:
@list |> map surface_wave_ms |> p
```

susceptible_to_infected

`susceptible_to_infected` — batch 61 builtin.

```
my $result = susceptible_to_infected $input
```

sw

`sw` — trivial string ops builtin. Alias for `starts_with`.

```
my $result = sw $x
# or in a pipeline:
@list |> map sw |> p
```

swamee_jain_factor

`swamee_jain_factor` — batch 13 builtin.

```
my $result = swamee_jain_factor $x
# or in a pipeline:
@list |> map swamee_jain_factor |> p
```

swampland_distance_check

`swampland_distance_check` — batch 39 builtin.

```
my $result = swampland_distance_check $x
# or in a pipeline:
@list |> map swampland_distance_check |> p
```

swap_bits

`swap_bits` — batch 10 builtin. Alias for `swap_bits_pos`.

```
my $result = swap_bits $x
# or in a pipeline:
@list |> map swap_bits |> p
```

swap_bits_pos

`swap_bits_pos` — batch 10 builtin.

```
my $result = swap_bits_pos $x
# or in a pipeline:
@list |> map swap_bits_pos |> p
```

swap_case

`swap_case` — trivial string ops builtin.

```
my $result = swap_case $x
# or in a pipeline:
@list |> map swap_case |> p
```

swap_free

`swap_free` — free swap space in bytes.

```
p format_bytes(swap_free)
my $pct = swap_total > 0 ? int(swap_free / swap_total * 100) : 100
p "$pct% swap free"
```

swap_indices

`swap_indices` — batch 9 builtin.

```
my $result = swap_indices $x
# or in a pipeline:
@list |> map swap_indices |> p
```

swap_pairs

`swap_pairs` — collection helpers (trivial) builtin.

```
my $result = swap_pairs $x
# or in a pipeline:
@list |> map swap_pairs |> p
```

swap_total

`swap_total` — total swap space in bytes. Linux: `/proc/meminfo SwapTotal`. macOS: `vm.swapusage` sysctl. Returns 0 if swap is disabled, `undef` on unsupported platforms.

```
p swap_total                            # 8589934592
p format_bytes(swap_total)              # 8.0 GiB
```

swap_used

`swap_used` — used swap space in bytes (total - free).

```
p format_bytes(swap_used)
p "swap pressure" if swap_used > swap_total * 0.8
```

swave_velocity_depth

`swave_velocity_depth` — batch 65 builtin.

```
my $result = swave_velocity_depth $x
# or in a pipeline:
@list |> map swave_velocity_depth |> p
```

swelling_strain_step

`swelling_strain_step` — batch 38 builtin.

```
my $result = swelling_strain_step $x
# or in a pipeline:
@list |> map swelling_strain_step |> p
```

sym4_coeffs

`sym4_coeffs` — batches 14-16 builtin.

```
my $result = sym4_coeffs $x
# or in a pipeline:
@list |> map sym4_coeffs |> p
```

sym_check

`sym_check` — batch 67 builtin.

```
my $result = sym_check $x
# or in a pipeline:
@list |> map sym_check |> p
```

sym_links

`sym_links` — filesystem extensions builtin.

```
my $result = sym_links $x
# or in a pipeline:
@list |> map sym_links |> p
```

symmetric_diff

`symmetric_diff` — collection (batch 2) builtin.

```
my $result = symmetric_diff $x
# or in a pipeline:
@list |> map symmetric_diff |> p
```

symplectic_euler_step

`symplectic_euler_step` — batch 34 builtin.

```
my $result = symplectic_euler_step $x
# or in a pipeline:
@list |> map symplectic_euler_step |> p
```

synge_world_function

`synge_world_function` — batch 39 builtin.

```
my $result = synge_world_function $x
# or in a pipeline:
@list |> map synge_world_function |> p
```

synodic_period

`synodic_period` — batch 12 builtin.

```
my $result = synodic_period $x
# or in a pipeline:
@list |> map synodic_period |> p
```

sys_uptime

`sys_uptime` — system uptime in seconds (float). On Linux reads `/proc/uptime`. On macOS uses `kern.boottime` sysctl. This is the wall-clock uptime of the machine, not the stryke process (see `uptime_secs` for process uptime).

```
p sys_uptime                            # 147951.5
my $days = int(sys_uptime / 86400)
p "up $days days"
```

t1ha

`t1ha` — batch 81 builtin.

```
my $result = t1ha $x
# or in a pipeline:
@list |> map t1ha |> p
```

t_test_one_sample

`t_test_one_sample($mu, @sample)` (alias `ttest1`) — performs a one-sample t-test comparing a sample mean against a hypothesized population mean mu. Returns [t_statistic, degrees_of_freedom]. Use t-distribution tables for p-value lookup.

```
my @sample = (5.1, 4.9, 5.2, 5.0, 4.8)
my $result = ttest1(5.0, @sample)
p "t=$result->[0] df=$result->[1]"
```

t_test_two_sample

`t_test_two_sample(\@a, \@b)` (alias `ttest2`) — performs an independent two-sample t-test comparing means of two samples. Assumes equal variances. Returns [t_statistic, degrees_of_freedom].

```
my @a = (5.1, 5.3, 4.9)
my @b = (4.2, 4.5, 4.1)
my $r = ttest2(\@a, \@b)
p "t=$r->[0] df=$r->[1]"
```

tableau_branch_close

`tableau_branch_close` — batch 67 builtin.

```
my $result = tableau_branch_close $x
# or in a pipeline:
@list |> map tableau_branch_close |> p
```

tabu_search_move_score

`tabu_search_move_score` — batch 41 builtin.

```
my $result = tabu_search_move_score $x
# or in a pipeline:
@list |> map tabu_search_move_score |> p
```

tabulate

`tabulate` — frequency table of values. Returns hash of value => count. Like R's table().

```
my %t = %{tabulate([qw(a b a c b a)])}
p $t{a}  # 3
```

tackle_efficiency

`tackle_efficiency` — batch 56 builtin.

```
my $result = tackle_efficiency $x
# or in a pipeline:
@list |> map tackle_efficiency |> p
```

tafel_anodic_current

`tafel_anodic_current` — batch 38 builtin.

```
my $result = tafel_anodic_current $x
# or in a pipeline:
@list |> map tafel_anodic_current |> p
```

tafel_cathodic_current

`tafel_cathodic_current` — batch 38 builtin.

```
my $result = tafel_cathodic_current $x
# or in a pipeline:
@list |> map tafel_cathodic_current |> p
```

tail_lines

`tail_lines` — file stat / path builtin.

```
my $result = tail_lines $x
# or in a pipeline:
@list |> map tail_lines |> p
```

tail_n

`tail_n` — list helpers (batch 4) builtin.

```
my $result = tail_n $x
# or in a pipeline:
@list |> map tail_n |> p
```

tail_str

`tail_str` — string (batch 2) builtin. Alias for `right_str`.

```
my $result = tail_str $x
# or in a pipeline:
@list |> map tail_str |> p
```

tails

`tails` — additional missing stdlib functions builtin.

```
my $result = tails $x
# or in a pipeline:
@list |> map tails |> p
```

tajimas_d

`tajimas_d` — batch 33 builtin.

```
my $result = tajimas_d $x
# or in a pipeline:
@list |> map tajimas_d |> p
```

take_every

`take_every` — list helpers (batch 4) builtin.

```
my $result = take_every $x
# or in a pipeline:
@list |> map take_every |> p
```

take_last

`take_last` — file stat / path builtin.

```
my $result = take_last $x
# or in a pipeline:
@list |> map take_last |> p
```

take_n

`take_n` — collection helpers (trivial) builtin.

```
my $result = take_n $x
# or in a pipeline:
@list |> map take_n |> p
```

talb

`talb` — ruby enumerable extras builtin. Alias for `tally_by`.

```
my $result = talb $x
# or in a pipeline:
@list |> map talb |> p
```

tally_by

`tally_by` — ruby enumerable extras builtin.

```
my $result = tally_by $x
# or in a pipeline:
@list |> map tally_by |> p
```

tan

`tan` — trig / math (batch 2) builtin.

```
my $result = tan $x
# or in a pipeline:
@list |> map tan |> p
```

tanh

`tanh` — trig / math (batch 2) builtin.

```
my $result = tanh $x
# or in a pipeline:
@list |> map tanh |> p
```

tanh_grad

`tanh_grad` — batch 27 builtin.

```
my $result = tanh_grad $x
# or in a pipeline:
@list |> map tanh_grad |> p
```

tanimoto_bits

`tanimoto_bits` — batch 33 builtin.

```
my $result = tanimoto_bits $input
```

tap_debug

`tap_debug` — conversion / utility (batch 4) builtin.

```
my $result = tap_debug $x
# or in a pipeline:
@list |> map tap_debug |> p
```

tap_val

`tap_val` — functional combinators builtin.

```
my $result = tap_val $x
# or in a pipeline:
@list |> map tap_val |> p
```

tapply

`tapply` — apply a function by group. Takes data, group labels, and function. Returns hash. Like R's tapply().

```
my %means = %{tapply([1,2,3,4], ["a","a","b","b"], fn { avg(@{$_[0]}) })}
p $means{a}  # 1.5
p $means{b}  # 3.5
```

tar_header_checksum

`tar_header_checksum` — batch 62 builtin.

```
my $result = tar_header_checksum $x
# or in a pipeline:
@list |> map tar_header_checksum |> p
```

tar_member_record

`tar_member_record` — batch 62 builtin.

```
my $result = tar_member_record $x
# or in a pipeline:
@list |> map tar_member_record |> p
```

tar_pad_512

`tar_pad_512` — batch 62 builtin.

```
my $result = tar_pad_512 $x
# or in a pipeline:
@list |> map tar_pad_512 |> p
```

target_heart_rate

`target_heart_rate` — batch 12 builtin.

```
my $result = target_heart_rate $x
# or in a pipeline:
@list |> map target_heart_rate |> p
```

tarjan_scc_step

`tarjan_scc_step` — batch 75 builtin.

```
my $result = tarjan_scc_step $x
# or in a pipeline:
@list |> map tarjan_scc_step |> p
```

tau

`tau` — constants builtin.

```
my $result = tau $x
# or in a pipeline:
@list |> map tau |> p
```

tax

`tax` — finance builtin.

```
my $result = tax $x
# or in a pipeline:
@list |> map tax |> p
```

tax_amount

`tax_amount` — physics formulas builtin.

```
my $result = tax_amount $x
# or in a pipeline:
@list |> map tax_amount |> p
```

tax_incidence

tax_incidence — batch 73 builtin.

```
my $result = tax_incidence $x
# or in a pipeline:
@list |> map tax_incidence |> p
```

taylor_diagram_score

taylor_diagram_score — batch 57 builtin.

```
my $result = taylor_diagram_score $x
# or in a pipeline:
@list |> map taylor_diagram_score |> p
```

taylor_microscale_step

taylor_microscale_step — batch 42 builtin.

```
my $result = taylor_microscale_step $x
# or in a pipeline:
@list |> map taylor_microscale_step |> p
```

taylor_window

taylor_window — batch 74 builtin.

```
my $result = taylor_window $x
# or in a pipeline:
@list |> map taylor_window |> p
```

tcrossprod

tcrossprod — M * t(M) (R's tcrossprod).

```
my $aat = tcrossprod([[1,2],[3,4]])  # [[5,11],[11,25]]
```

td

td — serialization (stryke-only encoders) builtin. Alias for toml_decode.

```
my $result = td $x
# or in a pipeline:
@list |> map td |> p
```

td_lambda_update

td_lambda_update — batch 40 builtin.

```
my $result = td_lambda_update $x
# or in a pipeline:
@list |> map td_lambda_update |> p
```

td_zero_update

td_zero_update — batch 40 builtin.

```
my $result = td_zero_update $x
# or in a pipeline:
@list |> map td_zero_update |> p
```

tdilate

`time_dilation($proper_time, $velocity)` (alias `tdilate`) — dilated time $\Delta t = \Delta t_0 \times \gamma$. Time appears to slow for moving objects.

```
p tdilate(1, 0.99 * 3e8)   # ~7.09 s (1 second at 99% c)
```

te

`te` — serialization (stryke-only encoders) builtin. Alias for `toml_encode`.

```
my $result = te $x
# or in a pipeline:
@list |> map te |> p
```

tee_iter

`tee_iter` — python/ruby stdlib builtin.

```
my $result = tee_iter $x
# or in a pipeline:
@list |> map tee_iter |> p
```

teei

`teei` — python/ruby stdlib builtin. Alias for `tee_iter`.

```
my $result = teei $x
# or in a pipeline:
@list |> map teei |> p
```

temp_dir

`temp_dir` — system introspection builtin.

```
my $result = temp_dir $x
# or in a pipeline:
@list |> map temp_dir |> p
```

template_match

`template_match` — batch 77 builtin.

```
my $result = template_match $x
# or in a pipeline:
@list |> map template_match |> p
```

tempo_to_ms

`tempo_to_ms` — batch 64 builtin.

```
my $result = tempo_to_ms $input
```

tempplanck

`tempplanck` — physics constants builtin. Alias for `planck_temperature`.

```
my $result = tempplanck $x
# or in a pipeline:
@list |> map tempplanck |> p
```

tensor_antisymmetrize_two

`tensor_antisymmetrize_two` — batch 39 builtin.

```
my $result = tensor_antisymmetrize_two $x
# or in a pipeline:
@list |> map tensor_antisymmetrize_two |> p
```

tensor_contract_two

`tensor_contract_two` — batch 39 builtin.

```
my $result = tensor_contract_two $x
# or in a pipeline:
@list |> map tensor_contract_two |> p
```

tensor_outer_two

`tensor_outer_two` — batch 39 builtin.

```
my $result = tensor_outer_two $x
# or in a pipeline:
@list |> map tensor_outer_two |> p
```

tensor_symmetrize_two

`tensor_symmetrize_two` — batch 39 builtin.

```
my $result = tensor_symmetrize_two $x
# or in a pipeline:
@list |> map tensor_symmetrize_two |> p
```

tensor_trace_index

`tensor_trace_index` — batch 39 builtin.

```
my $result = tensor_trace_index $x
# or in a pipeline:
@list |> map tensor_trace_index |> p
```

term_life_a_n_t

`term_life_a_n_t` — batch 60 builtin.

```
my $result = term_life_a_n_t $x
# or in a pipeline:
@list |> map term_life_a_n_t |> p
```

terminal_velocity

`terminal_velocity` — batch 12 builtin.

```
my $result = terminal_velocity $x
# or in a pipeline:
@list |> map terminal_velocity |> p
```

test_skip

`test_skip` — testing framework builtin.

```
my $result = test_skip $x
# or in a pipeline:
@list |> map test_skip |> p
```

tetrahedral_number

`tetrahedral_number` — batch 10 builtin.

```
my $result = tetrahedral_number $x
# or in a pipeline:
@list |> map tetrahedral_number |> p
```

text_after

`text_after` — extended stdlib builtin.

```
my $result = text_after $x
# or in a pipeline:
@list |> map text_after |> p
```

text_after_last

`text_after_last` — extended stdlib builtin.

```
my $result = text_after_last $x
# or in a pipeline:
@list |> map text_after_last |> p
```

text_before

`text_before` — extended stdlib builtin.

```
my $result = text_before $x
# or in a pipeline:
@list |> map text_before |> p
```

text_before_last

`text_before_last` — extended stdlib builtin.

```
my $result = text_before_last $x
# or in a pipeline:
@list |> map text_before_last |> p
```

text_between

`text_between` — extended stdlib builtin.

```
my $result = text_between $x
# or in a pipeline:
@list |> map text_between |> p
```

tf2sos_step

`tf2sos_step` — batch 74 builtin.

```
my $result = tf2sos_step $x
# or in a pipeline:
@list |> map tf2sos_step |> p
```

tf2zpk_step

`tf2zpk_step` — batch 74 builtin.

```
my $result = tf2zpk_step $x
# or in a pipeline:
@list |> map tf2zpk_step |> p
```

tfld

`tfld` — rust iterator methods builtin. Alias for `try_fold`.

```
my $result = tfld $x
# or in a pipeline:
@list |> map tfld |> p
```

theil_sen_slope

`theil_sen_slope` — batches 14-16 builtin.

```
my $result = theil_sen_slope $x
# or in a pipeline:
@list |> map theil_sen_slope |> p
```

then_fn

`then_fn` — python/ruby stdlib builtin.

```
my $result = then_fn $x
# or in a pipeline:
@list |> map then_fn |> p
```

thermal_mean_photons

`thermal_mean_photons` — batch 32 builtin.

```
my $result = thermal_mean_photons $x
# or in a pipeline:
@list |> map thermal_mean_photons |> p
```

thermal_runaway_threshold

`thermal_runaway_threshold` — batch 38 builtin.

```
my $result = thermal_runaway_threshold $x
# or in a pipeline:
@list |> map thermal_runaway_threshold |> p
```

thermal_wind_step

`thermal_wind_step` — batch 42 builtin.

```
my $result = thermal_wind_step $x
# or in a pipeline:
@list |> map thermal_wind_step |> p
```

theta2

`theta2` — batch 29 builtin.

```
my $result = theta2 $x
# or in a pipeline:
@list |> map theta2 |> p
```

theta3

`theta3` — batch 29 builtin.

```
my $result = theta3 $x
# or in a pipeline:
@list |> map theta3 |> p
```

thfn

`thfn` — python/ruby stdlib builtin. Alias for `then_fn`.

```
my $result = thfn $x
# or in a pipeline:
@list |> map thfn |> p
```

thin_lens

`thin_lens($object_dist, $focal_length)` (alias `thinlens`) — image distance using 1/f = 1/do + 1/di.

```
p thinlens(30, 20)   # 60 cm (real image)
p thinlens(10, 20)   # -20 cm (virtual image)
```

thin_lens_v

`thin_lens_v` — batch 23 builtin.

```
my $result = thin_lens_v $x
# or in a pipeline:
@list |> map thin_lens_v |> p
```

third_elem

`third_elem` — list helpers (batch 4) builtin.

```
my $result = third_elem $x
# or in a pipeline:
@list |> map third_elem |> p
```

thompson_sample_beta

`thompson_sample_beta` — batch 40 builtin.

```
my $result = thompson_sample_beta $x
# or in a pipeline:
@list |> map thompson_sample_beta |> p
```

three_opt_delta

`three_opt_delta` — batch 41 builtin.

```
my $result = three_opt_delta $x
# or in a pipeline:
@list |> map three_opt_delta |> p
```

threshold_act

`threshold_act` — batch 27 builtin.

```
my $result = threshold_act $x
# or in a pipeline:
@list |> map threshold_act |> p
```

threshold_hard_value

threshold_hard_value — batch 40 builtin.

```
my $result = threshold_hard_value $x
# or in a pipeline:
@list |> map threshold_hard_value |> p
```

threshold_soft_value

threshold_soft_value — batch 40 builtin.

```
my $result = threshold_soft_value $x
# or in a pipeline:
@list |> map threshold_soft_value |> p
```

thue_morse_string

thue_morse_string — batches 14-16 builtin.

```
my $result = thue_morse_string $x
# or in a pipeline:
@list |> map thue_morse_string |> p
```

ti2

ti2 — batch 29 builtin.

```
my $result = ti2 $x
# or in a pipeline:
@list |> map ti2 |> p
```

tidal_force

tidal_force — batch 31 builtin.

```
my $result = tidal_force $x
# or in a pipeline:
@list |> map tidal_force |> p
```

tied

tied — type / reflection builtin.

```
my $result = tied $x
# or in a pipeline:
@list |> map tied |> p
```

tile_xyz_to_lat_lng

tile_xyz_to_lat_lng — batch 58 builtin.

```
my $result = tile_xyz_to_lat_lng $input
```

time_sig_subdivision

time_sig_subdivision — batch 64 builtin.

```
my $result = time_sig_subdivision $x
# or in a pipeline:
@list |> map time_sig_subdivision |> p
```

times_fn

`times_fn` — python/ruby stdlib builtin.

```
my $result = times_fn $x
# or in a pipeline:
@list |> map times_fn |> p
```

timf

`timf` — python/ruby stdlib builtin. Alias for `times_fn`.

```
my $result = timf $x
# or in a pipeline:
@list |> map timf |> p
```

tip

`tip` — finance builtin.

```
my $result = tip $x
# or in a pipeline:
@list |> map tip |> p
```

tip_amount

`tip_amount` — physics formulas builtin.

```
my $result = tip_amount $x
# or in a pipeline:
@list |> map tip_amount |> p
```

tisserand_param

`tisserand_param` — batch 55 builtin.

```
my $result = tisserand_param $x
# or in a pipeline:
@list |> map tisserand_param |> p
```

tit_for_tat_step

`tit_for_tat_step` — batch 43 builtin.

```
my $result = tit_for_tat_step $x
# or in a pipeline:
@list |> map tit_for_tat_step |> p
```

title

`title` — trivial string ops builtin. Alias for `title_case`.

```
my $result = title $x
# or in a pipeline:
@list |> map title |> p
```

title_case

`title_case` — trivial string ops builtin.

```
my $result = title_case $x
# or in a pipeline:
@list |> map title_case |> p
```

title_words

`title_words` — batch 9 builtin. Alias for `capitalize_words`.

```
my $result = title_words $x
# or in a pipeline:
@list |> map title_words |> p
```

titration_endpoint_index

`titration_endpoint_index` — batch 38 builtin.

```
my $result = titration_endpoint_index $x
# or in a pipeline:
@list |> map titration_endpoint_index |> p
```

titration_ph_endpoint

`titration_ph_endpoint` — batch 63 builtin.

```
my $result = titration_ph_endpoint $x
# or in a pipeline:
@list |> map titration_ph_endpoint |> p
```

titration_volume

`titration_volume` — batch 12 builtin.

```
my $result = titration_volume $x
# or in a pipeline:
@list |> map titration_volume |> p
```

tkeys

`tkeys` — python/ruby stdlib builtin. Alias for `transform_keys`.

```
my $result = tkeys $x
# or in a pipeline:
@list |> map tkeys |> p
```

tle_mean_motion

`tle_mean_motion` — batch 55 builtin.

```
my $result = tle_mean_motion $x
# or in a pipeline:
@list |> map tle_mean_motion |> p
```

tm_marmur

`tm_marmur` — batch 33 builtin.

```
my $result = tm_marmur $x
# or in a pipeline:
@list |> map tm_marmur |> p
```

tm_wallace

`tm_wallace` — batch 33 builtin.

```
my $result = tm_wallace $x
# or in a pipeline:
@list |> map tm_wallace |> p
```

to_array

`to_array` — list helpers (batch 4) builtin.

```
my $result = to_array $input
```

to_ascii

`to_ascii` — extended stdlib builtin.

```
my $result = to_ascii $input
```

to_base

`to_base` — base conversion builtin.

```
my $result = to_base $input
```

to_bin

`to_bin` — base conversion builtin.

```
my $result = to_bin $input
```

to_bool

`to_bool` — conversion / utility (batch 4) builtin.

```
my $result = to_bool $input
```

to_csv_line

`to_csv_line` — string helpers (batch 4) builtin.

```
my $result = to_csv_line $input
```

to_emoji_num

`to_emoji_num` — encoding / phonetics builtin.

```
my $result = to_emoji_num $input
```

to_float

`to_float` — conversion / utility (batch 4) builtin.

```
my $result = to_float $input
```

to_float_each

`to_float_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = to_float_each $input
```

to_hash

`to_hash LIST` — collect a flat list of key-value pairs into a Perl hash.

The list is consumed two elements at a time: odd-positioned elements become keys and even-positioned elements become values. If there is an odd number of elements, the last key maps to `undef`. This is a terminal operation that materializes the full stream. Useful for converting pipeline output into a lookup table.

```
my %h = qw(a 1 b 2) |> to_hash
@pairs |> to_hash
my %freq = @words |> map { _, 1 } |> to_hash
```

to_hex

`to_hex` — base conversion builtin.

```
my $result = to_hex $input
```

to_int

`to_int` — conversion / utility (batch 4) builtin.

```
my $result = to_int $input
```

to_int_each

`to_int_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = to_int_each $input
```

to_oct

`to_oct` — base conversion builtin.

```
my $result = to_oct $input
```

to_pairs

`to_pairs` — list helpers (batch 4) builtin.

```
my $result = to_pairs $input
```

to_reversed

`to_reversed` — javascript array/object methods builtin.

```
my $result = to_reversed $input
```

to_set

`to_set LIST` — collect a list or iterator into a `set` object with O(1) membership testing.

The resulting set contains only unique elements (duplicates are discarded). This is a terminal operation that materializes the full stream. The returned set supports `->contains($val)`, `->union($other)`, `->intersection($other)`, and `->difference($other)`. Use this when you need fast repeated lookups against a collection of values.

```
my $s = 1:5 |> to_set
@words |> to_set                    # deduplicated set
my $allowed = to_set @whitelist
p $allowed->contains("foo")
```

to_sorted

`to_sorted` — javascript array/object methods builtin.

```
my $result = to_sorted $input
```

to_spliced

`to_spliced` — javascript array/object methods builtin.

```
my $result = to_spliced $input
```

to_string

`to_string` — conversion / utility (batch 4) builtin.

```
my $result = to_string $input
```

to_string_val

`to_string_val` — misc / utility builtin.

```
my $result = to_string_val $input
```

toasc

`toasc` — extended stdlib builtin. Alias for `to_ascii`.

```
my $result = toasc $x
# or in a pipeline:
@list |> map toasc |> p
```

tobit_log_likelihood

`tobit_log_likelihood` — batch 36 builtin.

```
my $result = tobit_log_likelihood $x
# or in a pipeline:
@list |> map tobit_log_likelihood |> p
```

today

`today` — date (batch 2) builtin.

```
my $result = today $x
# or in a pipeline:
@list |> map today |> p
```

todd_genus_eval

`todd_genus_eval` — batch 37 builtin.

```
my $result = todd_genus_eval $x
# or in a pipeline:
@list |> map todd_genus_eval |> p
```

toffoli_gate

`toffoli_gate` — batches 14-16 builtin.

```
my $result = toffoli_gate $x
# or in a pipeline:
@list |> map toffoli_gate |> p
```

token

`token` — id helpers builtin.

```
my $result = token $x
# or in a pipeline:
@list |> map token |> p
```

tomorrow

`tomorrow` — date (batch 2) builtin.

```
my $result = tomorrow $x
# or in a pipeline:
@list |> map tomorrow |> p
```

top

`top` — pipeline / string helpers builtin.

```
my $result = top $x
# or in a pipeline:
@list |> map top |> p
```

top_trading_cycle

`top_trading_cycle` — batch 73 builtin.

```
my $result = top_trading_cycle $x
# or in a pipeline:
@list |> map top_trading_cycle |> p
```

top_trading_cycles_step

`top_trading_cycles_step` — batch 43 builtin.

```
my $result = top_trading_cycles_step $x
# or in a pipeline:
@list |> map top_trading_cycles_step |> p
```

topk_indices

`topk_indices` — batch 27 builtin.

```
my $result = topk_indices $x
# or in a pipeline:
@list |> map topk_indices |> p
```

topo_kahn_step

`topo_kahn_step` — batch 75 builtin.

```
my $result = topo_kahn_step $x
# or in a pipeline:
@list |> map topo_kahn_step |> p
```

topological_sort

`topological_sort` — extended stdlib builtin.

```
my $result = topological_sort $x
# or in a pipeline:
@list |> map topological_sort |> p
```

topological_sort_kahn

`topological_sort_kahn` — batch 24 builtin.

```
my $result = topological_sort_kahn $x
# or in a pipeline:
@list |> map topological_sort_kahn |> p
```

toposort

`toposort` — extended stdlib builtin. Alias for `topological_sort`.

```
my $result = toposort $x
# or in a pipeline:
@list |> map toposort |> p
```

torque

`torque($force, $lever_arm, $angle)` — compute torque τ = r×F×sin(θ) (N□m). Angle in degrees, default 90.

```
p torque(50, 0.5)     # 25 N·m (perpendicular force)
p torque(50, 0.5, 30) # 12.5 N·m
```

torsion_coefficient

`torsion_coefficient` — batch 37 builtin.

```
my $result = torsion_coefficient $x
# or in a pipeline:
@list |> map torsion_coefficient |> p
```

tortuosity_estimate_bruggeman

`tortuosity_estimate_bruggeman` — batch 38 builtin.

```
my $result = tortuosity_estimate_bruggeman $x
# or in a pipeline:
@list |> map tortuosity_estimate_bruggeman |> p
```

torus_surface

`torus_surface($major_r, $minor_r)` — computes the surface area of a torus. Returns `4π²Rr`.

```
p torus_surface(10, 2)   # ~789.6
```

torus_volume

`torus_volume($major_r, $minor_r)` — computes the volume of a torus with major radius R (center to tube center) and minor radius r (tube radius). Returns $2\pi^2 R r^2$.

```
p torus_volume(10, 2)   # ~789.6
```

torussurf

`torussurf` — geometry (extended) builtin. Alias for `torus_surface`.

```
my $result = torussurf $x
# or in a pipeline:
@list |> map torussurf |> p
```

torusvol

`torusvol` — geometry (extended) builtin. Alias for `torus_volume`.

```
my $result = torusvol $x
# or in a pipeline:
@list |> map torusvol |> p
```

total_solar_irradiance_step

`total_solar_irradiance_step` — batch 42 builtin.

```
my $result = total_solar_irradiance_step $x
# or in a pipeline:
@list |> map total_solar_irradiance_step |> p
```

total_variation_distance

`total_variation_distance` — batch 40 builtin.

```
my $result = total_variation_distance $x
# or in a pipeline:
@list |> map total_variation_distance |> p
```

totient_sum

`totient_sum` — math / numeric (uncategorized batch) builtin.

```
my $result = totient_sum $x
# or in a pipeline:
@list |> map totient_sum |> p
```

totsum

`totsum` — math / numeric (uncategorized batch) builtin. Alias for `totient_sum`.

```
my $result = totsum $x
# or in a pipeline:
@list |> map totsum |> p
```

touch

`touch` — filesystem extensions builtin.

```
my $result = touch $x
# or in a pipeline:
@list |> map touch |> p
```

tov_mass_limit

`tov_mass_limit` — batch 31 builtin.

```
my $result = tov_mass_limit $x
# or in a pipeline:
@list |> map tov_mass_limit |> p
```

tower_of_hanoi

`tower_of_hanoi` — algorithms / puzzles builtin.

```
my $result = tower_of_hanoi $x
# or in a pipeline:
@list |> map tower_of_hanoi |> p
```

tplanck

`tplanck` — physics constants builtin. Alias for `planck_time`.

```
my $result = tplanck $x
# or in a pipeline:
@list |> map tplanck |> p
```

trace_distance

`trace_distance` — batches 14-16 builtin.

```
my $result = trace_distance $x
# or in a pipeline:
@list |> map trace_distance |> p
```

trade_wind_speed

`trade_wind_speed` — batch 42 builtin.

```
my $result = trade_wind_speed $x
# or in a pipeline:
@list |> map trade_wind_speed |> p
```

tragedy_commons_metric

`tragedy_commons_metric` — batch 43 builtin.

```
my $result = tragedy_commons_metric $x
# or in a pipeline:
@list |> map tragedy_commons_metric |> p
```

trailing_zeros

`trailing_zeros` — base conversion builtin.

```
my $result = trailing_zeros $x
# or in a pipeline:
@list |> map trailing_zeros |> p
```

trait

`trait` declares an interface that classes can implement via `impl`. Traits define method signatures that implementing classes must provide.

Declaration:

```
trait Printable {
    fn to_str          # required (no body)
    fn debug { ... }   # optional default impl
}
```

Implementation:

```
class Item impl Printable {
    name: Str
    fn to_str { $self->name }
}
```

Multiple traits:

```
class Widget impl Printable, Comparable {
    ...
}
```

Note: trait method bodies provide default implementations; classes can override them.

trans_check

`trans_check` — batch 67 builtin.

```
my $result = trans_check $x
# or in a pipeline:
@list |> map trans_check |> p
```

transcribe_dna_rna

`transcribe_dna_rna` — batch 33 builtin.

```
my $result = transcribe_dna_rna $x
# or in a pipeline:
@list |> map transcribe_dna_rna |> p
```

transform_keys

`transform_keys` — python/ruby stdlib builtin.

```
my $result = transform_keys $x
# or in a pipeline:
@list |> map transform_keys |> p
```

transform_values

`transform_values` — python/ruby stdlib builtin.

```
my $result = transform_values $x
# or in a pipeline:
@list |> map transform_values |> p
```

transition_arc_eager

`transition_arc_eager` — batch 69 builtin.

```
my $result = transition_arc_eager $x
# or in a pipeline:
@list |> map transition_arc_eager |> p
```

transition_arc_standard

`transition_arc_standard` — batch 69 builtin.

```
my $result = transition_arc_standard $x
# or in a pipeline:
@list |> map transition_arc_standard |> p
```

transitivity

`transitivity` — batch 75 builtin.

```
my $result = transitivity $x
# or in a pipeline:
@list |> map transitivity |> p
```

translate

`translate` — batch 70 builtin.

```
my $result = translate $x
# or in a pipeline:
@list |> map translate |> p
```

translate_dna

`translate_dna` — batch 33 builtin.

```
my $result = translate_dna $x
# or in a pipeline:
@list |> map translate_dna |> p
```

translate_point

`translate_point($x, $y, $dx, $dy)` — translates a 2D point by offset (dx, dy). Returns [$x+$dx, $y+$dy]. Basic vector addition.

```
my $p = translate_point(1, 2, 3, 4)
p @$p   # (4, 6)
```

transmission_pair_index

`transmission_pair_index` — batch 61 builtin.

```
my $result = transmission_pair_index $x
# or in a pipeline:
@list |> map transmission_pair_index |> p
```

transmittance

`transmittance` — batch 21 builtin.

```
my $result = transmittance $x
# or in a pipeline:
@list |> map transmittance |> p
```

transmittance_to_a

`transmittance_to_a` — batch 63 builtin.

```
my $result = transmittance_to_a $input
```

transpose

`transpose` — collection more builtin.

```
my $result = transpose $x
# or in a pipeline:
@list |> map transpose |> p
```

transpose_axis

`transpose_axis` — batch 54 builtin.

```
my $result = transpose_axis $x
# or in a pipeline:
@list |> map transpose_axis |> p
```

transpose_freq_semitones

`transpose_freq_semitones` — batch 9 builtin.

```
my $result = transpose_freq_semitones $x
# or in a pipeline:
@list |> map transpose_freq_semitones |> p
```

transpose_semi

`transpose_semi` — batch 9 builtin. Alias for `transpose_freq_semitones`.

```
my $result = transpose_semi $x
# or in a pipeline:
@list |> map transpose_semi |> p
```

transpt

`transpt` — geometry (extended) builtin. Alias for `translate_point`.

```
my $result = transpt $x
# or in a pipeline:
@list |> map transpt |> p
```

trapezoid

`trapz` (alias `trapezoid`) integrates evenly-spaced samples using the trapezoidal rule. Optional dx (default 1).

```
my @y = map { _ ** 2 } 0:100
p trapz(\@y, 0.01)  # ≈ 0.3333
```

trapezoidal_lte

`trapezoidal_lte` — batch 34 builtin.

```
my $result = trapezoidal_lte $x
# or in a pipeline:
@list |> map trapezoidal_lte |> p
```

trapped_surface_check

`trapped_surface_check` — batch 39 builtin.

```
my $result = trapped_surface_check $x
# or in a pipeline:
@list |> map trapped_surface_check |> p
```

trasatti_voltammetry_charge

`trasatti_voltammetry_charge` — batch 38 builtin.

```
my $result = trasatti_voltammetry_charge $x
# or in a pipeline:
@list |> map trasatti_voltammetry_charge |> p
```

tree_kernel_collins

`tree_kernel_collins` — batch 69 builtin.

```
my $result = tree_kernel_collins $x
# or in a pipeline:
@list |> map tree_kernel_collins |> p
```

trev

`trev` — javascript array/object methods builtin. Alias for `to_reversed`.

```
my $result = trev $x
# or in a pipeline:
@list |> map trev |> p
```

treynor_ratio

`treynor_ratio` — batch 13 builtin.

```
my $result = treynor_ratio $x
# or in a pipeline:
@list |> map treynor_ratio |> p
```

trial_smallest_factor

`trial_smallest_factor` — batch 35 builtin.

```
my $result = trial_smallest_factor $x
# or in a pipeline:
@list |> map trial_smallest_factor |> p
```

triangle_area_heron

`triangle_area_heron` — batch 28 builtin.

```
my $result = triangle_area_heron $x
# or in a pipeline:
@list |> map triangle_area_heron |> p
```

triangle_area_pts

`triangle_area_pts` — batch 28 builtin.

```
my $result = triangle_area_pts $x
# or in a pipeline:
@list |> map triangle_area_pts |> p
```

triangle_circumradius

`triangle_circumradius` — batch 28 builtin.

```
my $result = triangle_circumradius $x
# or in a pipeline:
@list |> map triangle_circumradius |> p
```

triangle_hypotenuse

`triangle_hypotenuse` — geometry / physics builtin.

```
my $result = triangle_hypotenuse $x
# or in a pipeline:
@list |> map triangle_hypotenuse |> p
```

triangle_inradius

`triangle_inradius` — batch 28 builtin.

```
my $result = triangle_inradius $x
# or in a pipeline:
@list |> map triangle_inradius |> p
```

triangles_count

`triangles_count` — batch 75 builtin.

```
my $result = triangles_count $x
# or in a pipeline:
@list |> map triangles_count |> p
```

triangular_number

`triangular_number` — number theory / primes builtin.

```
my $result = triangular_number $x
# or in a pipeline:
@list |> map triangular_number |> p
```

triangular_seq

`triangular_seq` — sequences builtin.

```
my $result = triangular_seq $x
# or in a pipeline:
@list |> map triangular_seq |> p
```

trib

`trib` — math / numeric (uncategorized batch) builtin. Alias for `tribonacci`.

```
my $result = trib $x
# or in a pipeline:
@list |> map trib |> p
```

tribonacci

`tribonacci` — math / numeric (uncategorized batch) builtin.

```
my $result = tribonacci $x
# or in a pipeline:
@list |> map tribonacci |> p
```

trigamma

`trigamma` — batch 72 builtin.

```
my $result = trigamma $x
# or in a pipeline:
@list |> map trigamma |> p
```

trigger_strategy_payoff

`trigger_strategy_payoff` — batch 43 builtin.

```
my $result = trigger_strategy_payoff $x
# or in a pipeline:
@list |> map trigger_strategy_payoff |> p
```

trigram

`trigram` — string processing (uncategorized batch) builtin. Alias for `trigrams`.

```
my $result = trigram $x
# or in a pipeline:
@list |> map trigram |> p
```

trigram_perplexity

`trigram_perplexity` — batch 69 builtin.

```
my $result = trigram_perplexity $x
# or in a pipeline:
@list |> map trigram_perplexity |> p
```

trigrams

`trigrams` — string processing (uncategorized batch) builtin.

```
my $result = trigrams $x
# or in a pipeline:
@list |> map trigrams |> p
```

trim_each

`trim_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = trim_each $x
# or in a pipeline:
@list |> map trim_each |> p
```

trim_left

`trim_left` — string helpers (batch 4) builtin.

```
my $result = trim_left $x
# or in a pipeline:
@list |> map trim_left |> p
```

trim_right

`trim_right` — string helpers (batch 4) builtin.

```
my $result = trim_right $x
# or in a pipeline:
@list |> map trim_right |> p
```

trimmed_mean

`trimmed_mean` — list helpers (batch 4) builtin.

```
my $result = trimmed_mean $x
# or in a pipeline:
@list |> map trimmed_mean |> p
```

trinomial_call

`trinomial_call` — batch 20 builtin.

```
my $result = trinomial_call $x
# or in a pipeline:
@list |> map trinomial_call |> p
```

trinum

`trinum` — number theory / primes builtin. Alias for `triangular_number`.

```
my $result = trinum $x
# or in a pipeline:
@list |> map trinum |> p
```

triple

`triple` — trivial numeric / predicate builtins builtin.

```
my $result = triple $x
# or in a pipeline:
@list |> map triple |> p
```

trpo_kl_constraint

`trpo_kl_constraint` — batch 40 builtin.

```
my $result = trpo_kl_constraint $x
# or in a pipeline:
@list |> map trpo_kl_constraint |> p
```

trueskill_match_quality

`trueskill_match_quality` — batch 56 builtin.

```
my $result = trueskill_match_quality $x
# or in a pipeline:
@list |> map trueskill_match_quality |> p
```

trueskill_update

`trueskill_update` — batch 56 builtin.

```
my $result = trueskill_update $x
# or in a pipeline:
@list |> map trueskill_update |> p
```

trunc

`trunc` — trivial numeric / predicate builtins builtin.

```
my $result = trunc $x
# or in a pipeline:
@list |> map trunc |> p
```

truncate_at

`truncate_at` — trivial string ops builtin.

```
my $result = truncate_at $x
# or in a pipeline:
@list |> map truncate_at |> p
```

truncn

`truncn` — trivial numeric / predicate builtins builtin. Alias for `trunc`.

```
my $result = truncn $x
# or in a pipeline:
@list |> map truncn |> p
```

trust_game_repayment

`trust_game_repayment` — batch 43 builtin.

```
my $result = trust_game_repayment $x
# or in a pipeline:
@list |> map trust_game_repayment |> p
```

truth_table

`truth_table` — algorithms / puzzles builtin.

```
my $result = truth_table $x
# or in a pipeline:
@list |> map truth_table |> p
```

truthful_mechanism_check

`truthful_mechanism_check` — batch 43 builtin.

```
my $result = truthful_mechanism_check $x
# or in a pipeline:
@list |> map truthful_mechanism_check |> p
```

truthy_count

`truthy_count` — counters (batch 4) builtin.

```
my $result = truthy_count $x
# or in a pipeline:
@list |> map truthy_count |> p
```

try_fn

`try_fn` — functional combinators builtin.

```
my $result = try_fn $x
# or in a pipeline:
@list |> map try_fn |> p
```

try_fold

`try_fold` — rust iterator methods builtin.

```
my $result = try_fold $x
# or in a pipeline:
@list |> map try_fold |> p
```

ts_headline

`ts_headline` — batch 70 builtin.

```
my $result = ts_headline $x
# or in a pipeline:
@list |> map ts_headline |> p
```

ts_match

`ts_match` — batch 70 builtin.

```
my $result = ts_match $x
# or in a pipeline:
@list |> map ts_match |> p
```

ts_rank

`ts_rank` — batch 70 builtin.

```
my $result = ts_rank $x
# or in a pipeline:
@list |> map ts_rank |> p
```

tsirelson_bound

`tsirelson_bound` — batch 32 builtin.

```
my $result = tsirelson_bound $x
# or in a pipeline:
@list |> map tsirelson_bound |> p
```

tsp_christofides

`tsp_christofides` — batch 75 builtin.

```
my $result = tsp_christofides $x
# or in a pipeline:
@list |> map tsp_christofides |> p
```

tsp_held_karp

`tsp_held_karp` — batch 75 builtin.

```
my $result = tsp_held_karp $x
# or in a pipeline:
@list |> map tsp_held_karp |> p
```

tsp_held_karp_step

`tsp_held_karp_step` — batch 41 builtin.

```
my $result = tsp_held_karp_step $x
# or in a pipeline:
@list |> map tsp_held_karp_step |> p
```

tsp_lower_bound_mst

`tsp_lower_bound_mst` — batch 41 builtin.

```
my $result = tsp_lower_bound_mst $x
# or in a pipeline:
@list |> map tsp_lower_bound_mst |> p
```

tsp_nn_step

`tsp_nn_step` — batch 75 builtin.

```
my $result = tsp_nn_step $x
# or in a pipeline:
@list |> map tsp_nn_step |> p
```

tspl

`tspl` — javascript array/object methods builtin. Alias for `to_spliced`.

```
my $result = tspl $x
# or in a pipeline:
@list |> map tspl |> p
```

tsrt

`tsrt` — javascript array/object methods builtin. Alias for `to_sorted`.

```
my $result = tsrt $x
# or in a pipeline:
@list |> map tsrt |> p
```

tstr

`tstr` — misc / utility builtin. Alias for `to_string_val`.

```
my $result = tstr $x
# or in a pipeline:
@list |> map tstr |> p
```

tt_to_tdb

`tt_to_tdb` — batch 55 builtin.

```
my $result = tt_to_tdb $input
```

ttable

`ttable` — algorithms / puzzles builtin. Alias for `truth_table`.

```
my $result = ttable $x
# or in a pipeline:
@list |> map ttable |> p
```

ttl

`ttl` — batch 71 builtin.

```
my $result = ttl $x
# or in a pipeline:
@list |> map ttl |> p
```

tukey_lambda_quantile

`tukey_lambda_quantile` — batches 17-18 builtin.

```
my $result = tukey_lambda_quantile $x
# or in a pipeline:
@list |> map tukey_lambda_quantile |> p
```

tukey_w

`tukey_w` — batch 74 builtin.

```
my $result = tukey_w $x
# or in a pipeline:
@list |> map tukey_w |> p
```

tukey_window

`tukey_window` — batch 25 builtin.

```
my $result = tukey_window $x
# or in a pipeline:
@list |> map tukey_window |> p
```

tunneling_prob

`tunneling_prob` — batch 32 builtin.

```
my $result = tunneling_prob $x
# or in a pipeline:
@list |> map tunneling_prob |> p
```

turaev_alexander_step

`turaev_alexander_step` — batch 37 builtin.

```
my $result = turaev_alexander_step $x
# or in a pipeline:
@list |> map turaev_alexander_step |> p
```

turbulence_intensity_step

`turbulence_intensity_step` — batch 42 builtin.

```
my $result = turbulence_intensity_step $x
# or in a pipeline:
@list |> map turbulence_intensity_step |> p
```

turbulent_dissipation_eps

`turbulent_dissipation_eps` — batch 42 builtin.

```
my $result = turbulent_dissipation_eps $x
# or in a pipeline:
@list |> map turbulent_dissipation_eps |> p
```

turbulent_kinetic_energy_step

`turbulent_kinetic_energy_step` — batch 42 builtin.

```
my $result = turbulent_kinetic_energy_step $x
# or in a pipeline:
@list |> map turbulent_kinetic_energy_step |> p
```

tvals

`tvals` — python/ruby stdlib builtin. Alias for `transform_values`.

```
my $result = tvals $x
# or in a pipeline:
@list |> map tvals |> p
```

tversky

`tversky` — batch 27 builtin.

```
my $result = tversky $x
# or in a pipeline:
@list |> map tversky |> p
```

twin_primes

`twin_primes` — number theory / primes builtin.

```
my $result = twin_primes $x
# or in a pipeline:
@list |> map twin_primes |> p
```

twinp

`twinp` — number theory / primes builtin. Alias for `twin_primes`.

```
my $result = twinp $x
# or in a pipeline:
@list |> map twinp |> p
```

twist_tensor_step

`twist_tensor_step` — batch 39 builtin.

```
my $result = twist_tensor_step $x
# or in a pipeline:
@list |> map twist_tensor_step |> p
```

two_opt_swap_delta

`two_opt_swap_delta` — batch 41 builtin.

```
my $result = two_opt_swap_delta $x
# or in a pipeline:
@list |> map two_opt_swap_delta |> p
```

two_sls_iv

`two_sls_iv` — batch 36 builtin.

```
my $result = two_sls_iv $x
# or in a pipeline:
@list |> map two_sls_iv |> p
```

two_watched_literals

`two_watched_literals` — batch 67 builtin.

```
my $result = two_watched_literals $x
# or in a pipeline:
@list |> map two_watched_literals |> p
```

txaft

`txaft` — extended stdlib builtin. Alias for `text_after`.

```
my $result = txaft $x
# or in a pipeline:
@list |> map txaft |> p
```

txaftl

`txaftl` — extended stdlib builtin. Alias for `text_after_last`.

```
my $result = txaftl $x
# or in a pipeline:
@list |> map txaftl |> p
```

txbef

`txbef` — extended stdlib builtin. Alias for `text_before`.

```
my $result = txbef $x
# or in a pipeline:
@list |> map txbef |> p
```

txbefl

`txbefl` — extended stdlib builtin. Alias for `text_before_last`.

```
my $result = txbefl $x
# or in a pipeline:
@list |> map txbefl |> p
```

txbtwn

`txbtwn` — extended stdlib builtin. Alias for `text_between`.

```
my $result = txbtwn $x
# or in a pipeline:
@list |> map txbtwn |> p
```

type

`type` — misc / utility builtin. Alias for `type_of`.

```
my $result = type $x
# or in a pipeline:
@list |> map type |> p
```

type_each

`type_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = type_each $x
# or in a pipeline:
@list |> map type_each |> p
```

type_of

`type_of` — misc / utility builtin.

```
my $result = type_of $x
# or in a pipeline:
@list |> map type_of |> p
```

type_redis

`type_redis` — batch 71 builtin.

```
my $result = type_redis $x
# or in a pipeline:
@list |> map type_redis |> p
```

typeof

`typeof` — misc / utility builtin. Alias for `type_of`.

```
my $result = typeof $x
# or in a pipeline:
@list |> map typeof |> p
```

tz

`tz` — base conversion builtin. Alias for `trailing_zeros`.

```
my $result = tz $x
# or in a pipeline:
@list |> map tz |> p
```

ubound

`ubound` — extended stdlib builtin. Alias for `upper_bound`.

```
my $result = ubound $x
# or in a pipeline:
@list |> map ubound |> p
```

ucb1_action_value

`ucb1_action_value` — batch 40 builtin.

```
my $result = ucb1_action_value $x
# or in a pipeline:
@list |> map ucb1_action_value |> p
```

ucm_unobserved_components

`ucm_unobserved_components` — batch 78 builtin.

```
my $result = ucm_unobserved_components $x
# or in a pipeline:
@list |> map ucm_unobserved_components |> p
```

ud

`ud` — crypto / encoding builtin. Alias for `url_decode`.

```
my $result = ud $x
# or in a pipeline:
@list |> map ud |> p
```

ue

`ue` — crypto / encoding builtin. Alias for `url_encode`.

```
my $result = ue $x
# or in a pipeline:
@list |> map ue |> p
```

ugz

`ugz` — crypto / encoding builtin. Alias for `gunzip`.

```
my $result = ugz $x
# or in a pipeline:
@list |> map ugz |> p
```

uid

`uid` — system introspection builtin.

```
my $result = uid $x
# or in a pipeline:
@list |> map uid |> p
```

uin

`uin` — algebraic match builtin. Alias for `update_in`.

```
my $result = uin $x
# or in a pipeline:
@list |> map uin |> p
```

ultimatum_acceptance_prob

`ultimatum_acceptance_prob` — batch 43 builtin.

```
my $result = ultimatum_acceptance_prob $x
# or in a pipeline:
@list |> map ultimatum_acceptance_prob |> p
```

uncertainty_position

`uncertainty_position` — batch 32 builtin.

```
my $result = uncertainty_position $x
# or in a pipeline:
@list |> map uncertainty_position |> p
```

undef_count

undef_count — counters (batch 4) builtin.

```
my $result = undef_count $x
# or in a pipeline:
@list |> map undef_count |> p
```

underline

underline — color / ansi builtin. Alias for red.

```
my $result = underline $x
# or in a pipeline:
@list |> map underline |> p
```

unfold

unfold — functional combinators builtin.

```
my $result = unfold $x
# or in a pipeline:
@list |> map unfold |> p
```

unfoldr

unfoldr — haskell list functions builtin.

```
my $result = unfoldr $x
# or in a pipeline:
@list |> map unfoldr |> p
```

unfr

unfr — haskell list functions builtin. Alias for unfoldr.

```
my $result = unfr $x
# or in a pipeline:
@list |> map unfr |> p
```

unicode_to_native

unicode_to_native — list / aggregate builtin.

```
my $result = unicode_to_native $input
```

unification_robinson

unification_robinson — batch 67 builtin.

```
my $result = unification_robinson $x
# or in a pipeline:
@list |> map unification_robinson |> p
```

union_list

union_list — additional missing stdlib functions builtin.

```
my $result = union_list $x
# or in a pipeline:
@list |> map union_list |> p
```

unionl

`unionl` — additional missing stdlib functions builtin. Alias for `union_list`.

```
my $result = unionl $x
# or in a pipeline:
@list |> map unionl |> p
```

uniq_by

`uniq_by` — python/ruby stdlib builtin.

```
my $result = uniq_by $x
# or in a pipeline:
@list |> map uniq_by |> p
```

unique_count

`unique_count` — batch 72 builtin.

```
my $result = unique_count $x
# or in a pipeline:
@list |> map unique_count |> p
```

unique_count_of

`unique_count_of` — list helpers (batch 4) builtin.

```
my $result = unique_count_of $x
# or in a pipeline:
@list |> map unique_count_of |> p
```

unique_words

`unique_words` — string processing (uncategorized batch) builtin.

```
my $result = unique_words $x
# or in a pipeline:
@list |> map unique_words |> p
```

uniqw

`uniqw` — string processing (uncategorized batch) builtin. Alias for `unique_words`.

```
my $result = uniqw $x
# or in a pipeline:
@list |> map uniqw |> p
```

unit_vec

`unit_vec` — matrix / linear algebra builtin. Alias for `vec_normalize`.

```
my $result = unit_vec $x
# or in a pipeline:
@list |> map unit_vec |> p
```

unit_vector

`unit_vector` — batch 11 builtin.

```
my $result = unit_vector $x
# or in a pipeline:
@list |> map unit_vector |> p
```

universal_code_length

`universal_code_length` — batch 40 builtin.

```
my $result = universal_code_length $x
# or in a pipeline:
@list |> map universal_code_length |> p
```

universal_enveloping_dim

`universal_enveloping_dim` — batch 37 builtin.

```
my $result = universal_enveloping_dim $x
# or in a pipeline:
@list |> map universal_enveloping_dim |> p
```

unix_epoch

`unix_epoch` — now / timestamp builtin.

```
my $result = unix_epoch $x
# or in a pipeline:
@list |> map unix_epoch |> p
```

unix_epoch_ms

`unix_epoch_ms` — now / timestamp builtin.

```
my $result = unix_epoch_ms $x
# or in a pipeline:
@list |> map unix_epoch_ms |> p
```

unix_epoch_to_iso

`unix_epoch_to_iso` — batch 13 builtin.

```
my $result = unix_epoch_to_iso $input
```

unlines

`unlines` — go/general functional utilities builtin.

```
my $result = unlines $x
# or in a pipeline:
@list |> map unlines |> p
```

unlns

`unlns` — go/general functional utilities builtin. Alias for `unlines`.

```
my $result = unlns $x
# or in a pipeline:
@list |> map unlns |> p
```

unquote

`unquote` — string quote / escape builtin.

```
my $result = unquote $x
# or in a pipeline:
@list |> map unquote |> p
```

unruh_temperature_full

`unruh_temperature_full` — batch 39 builtin.

```
my $result = unruh_temperature_full $x
# or in a pipeline:
@list |> map unruh_temperature_full |> p
```

unscented_sigma_point

`unscented_sigma_point` — batch 40 builtin.

```
my $result = unscented_sigma_point $x
# or in a pipeline:
@list |> map unscented_sigma_point |> p
```

unsharp_mask_kernel

`unsharp_mask_kernel` — batches 14-16 builtin.

```
my $result = unsharp_mask_kernel $x
# or in a pipeline:
@list |> map unsharp_mask_kernel |> p
```

unsharp_mask_step

`unsharp_mask_step` — batch 40 builtin.

```
my $result = unsharp_mask_step $x
# or in a pipeline:
@list |> map unsharp_mask_step |> p
```

untie

`untie` — type / reflection builtin.

```
my $result = untie $x
# or in a pipeline:
@list |> map untie |> p
```

unwds

`unwds` — go/general functional utilities builtin. Alias for `unwords`.

```
my $result = unwds $x
# or in a pipeline:
@list |> map unwds |> p
```

unweaken

`unweaken` — list / aggregate builtin.

```
my $result = unweaken $x
# or in a pipeline:
@list |> map unweaken |> p
```

unwords

`unwords` — go/general functional utilities builtin.

```
my $result = unwords $x
# or in a pipeline:
@list |> map unwords |> p
```

unzip

`unzip` — collection more builtin.

```
my $result = unzip $x
# or in a pipeline:
@list |> map unzip |> p
```

unzip_archive

`unzip_archive` — crypto / encoding builtin. Alias for `zip_extract`.

```
my $result = unzip_archive $x
# or in a pipeline:
@list |> map unzip_archive |> p
```

unzip_pairs

`unzip_pairs` — go/general functional utilities builtin.

```
my $result = unzip_pairs $x
# or in a pipeline:
@list |> map unzip_pairs |> p
```

upc_check

`upc_check` — batch 82 builtin.

```
my $result = upc_check $x
# or in a pipeline:
@list |> map upc_check |> p
```

upcase_each

`upcase_each` — trivial numeric helpers (batch 4) builtin.

```
my $result = upcase_each $x
# or in a pipeline:
@list |> map upcase_each |> p
```

upda

`upda` — go/general functional utilities builtin. Alias for `update_at`.

```
my $result = upda $x
# or in a pipeline:
@list |> map upda |> p
```

update_at

`update_at` — go/general functional utilities builtin.

```
my $result = update_at $x
# or in a pipeline:
@list |> map update_at |> p
```

update_in

`update_in` — algebraic match builtin.

```
my $result = update_in $x
# or in a pipeline:
@list |> map update_in |> p
```

upfirdn_step

`upfirdn_step` — batch 74 builtin.

```
my $result = upfirdn_step $x
# or in a pipeline:
@list |> map upfirdn_step |> p
```

upper_bound

`upper_bound` — extended stdlib builtin.

```
my $result = upper_bound $x
# or in a pipeline:
@list |> map upper_bound |> p
```

upper_snake

`upper_snake` — string (batch 2) builtin. Alias for `constant_case`.

```
my $result = upper_snake $x
# or in a pipeline:
@list |> map upper_snake |> p
```

upper_triangular_q

`upper_triangular_q` — batches 17-18 builtin.

```
my $result = upper_triangular_q $x
# or in a pipeline:
@list |> map upper_triangular_q |> p
```

uppercase

`uppercase` — string (batch 2) builtin.

```
my $result = uppercase $x
# or in a pipeline:
@list |> map uppercase |> p
```

upsample

`upsample(\@signal, $factor)` (alias `interpolate`) — increases sample rate by inserting zeros and filtering. Factor must be positive integer.

```
my $upsampled = interpolate(\@signal, 4)  # 4x sample rate
```

uptime_secs

`uptime_secs` — more process / system builtin.

```
my $result = uptime_secs $x
# or in a pipeline:
@list |> map uptime_secs |> p
```

upto

`upto` — python/ruby stdlib builtin.

```
my $result = upto $x
# or in a pipeline:
@list |> map upto |> p
```

uqb

`uqb` — python/ruby stdlib builtin. Alias for `uniq_by`.

```
my $result = uqb $x
# or in a pipeline:
@list |> map uqb |> p
```

uri_normalize

`uri_normalize` — batch 81 builtin.

```
my $result = uri_normalize $x
# or in a pipeline:
@list |> map uri_normalize |> p
```

uri_resolve

`uri_resolve` — batch 81 builtin.

```
my $result = uri_resolve $x
# or in a pipeline:
@list |> map uri_resolve |> p
```

uri_template_expand

`uri_template_expand` — batch 81 builtin.

```
my $result = uri_template_expand $x
# or in a pipeline:
@list |> map uri_template_expand |> p
```

url_decode_form

`url_decode_form` — batch 81 builtin.

```
my $result = url_decode_form $x
# or in a pipeline:
@list |> map url_decode_form |> p
```

url_encode_form

`url_encode_form` — batch 81 builtin.

```
my $result = url_encode_form $x
# or in a pipeline:
@list |> map url_encode_form |> p
```

url_host

`url_host` — url / email parts builtin.

```
my $result = url_host $x
# or in a pipeline:
@list |> map url_host |> p
```

url_origin

`url_origin` — batch 81 builtin.

```
my $result = url_origin $x
# or in a pipeline:
@list |> map url_origin |> p
```

url_path

`url_path` — url / email parts builtin.

```
my $result = url_path $x
# or in a pipeline:
@list |> map url_path |> p
```

url_query

`url_query` — url / email parts builtin.

```
my $result = url_query $x
# or in a pipeline:
@list |> map url_query |> p
```

url_scheme

`url_scheme` — url / email parts builtin.

```
my $result = url_scheme $x
# or in a pipeline:
@list |> map url_scheme |> p
```

urld

`urld` — extended stdlib builtin.

```
my $result = urld $x
# or in a pipeline:
@list |> map urld |> p
```

urle

`urle` — extended stdlib builtin.

```
my $result = urle $x
# or in a pipeline:
@list |> map urle |> p
```

us_to_ns

us_to_ns — file stat / path builtin.

```
my $result = us_to_ns $input
```

usamp

usamp — dsp / signal (extended) builtin. Alias for upsample.

```
my $result = usamp $x
# or in a pipeline:
@list |> map usamp |> p
```

username

username — system introspection builtin.

```
my $result = username $x
# or in a pipeline:
@list |> map username |> p
```

utc

utc — date / time builtin. Alias for datetime_utc.

```
my $result = utc $x
# or in a pipeline:
@list |> map utc |> p
```

utci_simple

utci_simple — batch 12 builtin.

```
my $result = utci_simple $x
# or in a pipeline:
@list |> map utci_simple |> p
```

utilitarian_solution

utilitarian_solution — batch 43 builtin.

```
my $result = utilitarian_solution $x
# or in a pipeline:
@list |> map utilitarian_solution |> p
```

utm_from_lat_lng

utm_from_lat_lng — batch 58 builtin.

```
my $result = utm_from_lat_lng $x
# or in a pipeline:
@list |> map utm_from_lat_lng |> p
```

utm_to_lat_lng

utm_to_lat_lng — batch 58 builtin.

```
my $result = utm_to_lat_lng $input
```

uuencode_step

`uuencode_step` — batch 62 builtin.

```
my $result = uuencode_step $x
# or in a pipeline:
@list |> map uuencode_step |> p
```

uuid_v4

`uuid_v4` — id helpers builtin.

```
my $result = uuid_v4 $x
# or in a pipeline:
@list |> map uuid_v4 |> p
```

uzp

`uzp` — go/general functional utilities builtin. Alias for `unzip_pairs`.

```
my $result = uzp $x
# or in a pipeline:
@list |> map uzp |> p
```

uzst

`uzst` — crypto / encoding builtin. Alias for `zstd_decode`.

```
my $result = uzst $x
# or in a pipeline:
@list |> map uzst |> p
```

v_polynomial_eval

`v_polynomial_eval` — batch 37 builtin.

```
my $result = v_polynomial_eval $x
# or in a pipeline:
@list |> map v_polynomial_eval |> p
```

vaccination_coverage_required

`vaccination_coverage_required` — batch 61 builtin.

```
my $result = vaccination_coverage_required $x
# or in a pipeline:
@list |> map vaccination_coverage_required |> p
```

vacuum_energy_density

`vacuum_energy_density` — batch 31 builtin.

```
my $result = vacuum_energy_density $x
# or in a pipeline:
@list |> map vacuum_energy_density |> p
```

vacuum_permeability

`vacuum_permeability` — physics constants builtin.

```
my $result = vacuum_permeability $x
# or in a pipeline:
@list |> map vacuum_permeability |> p
```

value_iteration_step

`value_iteration_step` — batch 40 builtin.

```
my $result = value_iteration_step $x
# or in a pipeline:
@list |> map value_iteration_step |> p
```

values_at

`values_at` — python/ruby stdlib builtin.

```
my $result = values_at $x
# or in a pipeline:
@list |> map values_at |> p
```

values_sorted

`values_sorted` — hash ops (batch 2) builtin.

```
my $result = values_sorted $x
# or in a pipeline:
@list |> map values_sorted |> p
```

van_leer_limiter

`van_leer_limiter` — batch 34 builtin.

```
my $result = van_leer_limiter $x
# or in a pipeline:
@list |> map van_leer_limiter |> p
```

van_t_hoff_kp

`van_t_hoff_kp` — batch 63 builtin.

```
my $result = van_t_hoff_kp $x
# or in a pipeline:
@list |> map van_t_hoff_kp |> p
```

vang

`vang` — geometry (extended) builtin. Alias for `vector_angle`.

```
my $result = vang $x
# or in a pipeline:
@list |> map vang |> p
```

vant_hoff

`vant_hoff` — batch 21 builtin.

```
my $result = vant_hoff $x
# or in a pipeline:
@list |> map vant_hoff |> p
```

var_fit

var_fit — batch 78 builtin.

```
my $result = var_fit $x
# or in a pipeline:
@list |> map var_fit |> p
```

var_historical

var_historical — batches 17-18 builtin.

```
my $result = var_historical $x
# or in a pipeline:
@list |> map var_historical |> p
```

variance_decomposition

variance_decomposition — batch 36 builtin.

```
my $result = variance_decomposition $x
# or in a pipeline:
@list |> map variance_decomposition |> p
```

vary_match

vary_match — batch 81 builtin.

```
my $result = vary_match $x
# or in a pipeline:
@list |> map vary_match |> p
```

vasicek_bond

vasicek_bond — batch 20 builtin.

```
my $result = vasicek_bond $x
# or in a pipeline:
@list |> map vasicek_bond |> p
```

vat

vat — python/ruby stdlib builtin. Alias for values_at.

```
my $result = vat $x
# or in a pipeline:
@list |> map vat |> p
```

vcg_payment

vcg_payment — batch 73 builtin.

```
my $result = vcg_payment $x
# or in a pipeline:
@list |> map vcg_payment |> p
```

vcg_payment_step

vcg_payment_step — batch 43 builtin.

```
my $result = vcg_payment_step $x
# or in a pipeline:
@list |> map vcg_payment_step |> p
```

vcross

`vector_cross(\@a, \@b)` (alias `vcross`) — computes the cross product of two 3D vectors. Returns a 3-element arrayref perpendicular to both inputs.

```
my $c = vcross([1,0,0], [0,1,0])
p @$c   # (0, 0, 1)
```

vdot

`vector_dot(\@a, \@b)` (alias `vdot`) — computes the dot product of two vectors. Returns the scalar sum of element-wise products.

```
p vdot([1,2,3], [4,5,6])   # 32
```

vec_add

`vec_add` — matrix / linear algebra builtin.

```
my $result = vec_add $x
# or in a pipeline:
@list |> map vec_add |> p
```

vec_clj

`vec_clj` — algebraic match builtin.

```
my $result = vec_clj $x
# or in a pipeline:
@list |> map vec_clj |> p
```

vec_dot

`vec_dot` — batch 11 builtin.

```
my $result = vec_dot $x
# or in a pipeline:
@list |> map vec_dot |> p
```

vec_l2

`vec_l2` — batch 11 builtin. Alias for `l2_norm`.

```
my $result = vec_l2 $x
# or in a pipeline:
@list |> map vec_l2 |> p
```

vec_normalize

`vec_normalize` — matrix / linear algebra builtin.

```
my $result = vec_normalize $x
# or in a pipeline:
@list |> map vec_normalize |> p
```

vec_outer

`vec_outer` — batch 11 builtin. Alias for `outer_product`.

```
my $result = vec_outer $x
# or in a pipeline:
@list |> map vec_outer |> p
```

vec_scale

`vec_scale` — matrix / linear algebra builtin.

```
my $result = vec_scale $x
# or in a pipeline:
@list |> map vec_scale |> p
```

vec_set_value

`vec_set_value` — internal builtin.

```
my $result = vec_set_value $x
# or in a pipeline:
@list |> map vec_set_value |> p
```

vec_sub

`vec_sub` — matrix / linear algebra builtin.

```
my $result = vec_sub $x
# or in a pipeline:
@list |> map vec_sub |> p
```

vecc

`vecc` — algebraic match builtin. Alias for `vec_clj`.

```
my $result = vecc $x
# or in a pipeline:
@list |> map vecc |> p
```

vecm_alpha_beta

`vecm_alpha_beta` — batch 36 builtin.

```
my $result = vecm_alpha_beta $x
# or in a pipeline:
@list |> map vecm_alpha_beta |> p
```

vecm_fit

`vecm_fit` — batch 78 builtin.

```
my $result = vecm_fit $x
# or in a pipeline:
@list |> map vecm_fit |> p
```

vector_angle

`vector_angle(\@a, \@b)` (alias `vangle`) — computes the angle in radians between two vectors using the dot product formula.

```
p vangle([1,0], [0,1])   # ~1.5708 (π/2)
p vangle([1,0], [1,0])   # 0
```

vector_magnitude

`vector_magnitude(\@v)` (alias `vmag`) — computes the Euclidean length (L2 norm) of a vector.

```
p vmag([3, 4])   # 5
p vmag([1, 2, 2])   # 3
```

vector_normalize

`vector_normalize(\@v)` (alias `vnorm`) — returns a unit vector in the same direction. Divides by magnitude.

```
my $u = vnorm([3, 4])
p @$u   # (0.6, 0.8)
```

vector_project

`vector_project` — batch 11 builtin.

```
my $result = vector_project $x
# or in a pipeline:
@list |> map vector_project |> p
```

vector_reject

`vector_reject` — batch 11 builtin.

```
my $result = vector_reject $x
# or in a pipeline:
@list |> map vector_reject |> p
```

velocity_potential_step

`velocity_potential_step` — batch 42 builtin.

```
my $result = velocity_potential_step $x
# or in a pipeline:
@list |> map velocity_potential_step |> p
```

verlinde_count_step

`verlinde_count_step` — batch 37 builtin.

```
my $result = verlinde_count_step $x
# or in a pipeline:
@list |> map verlinde_count_step |> p
```

verma_character_step

`verma_character_step` — batch 37 builtin.

```
my $result = verma_character_step $x
# or in a pipeline:
@list |> map verma_character_step |> p
```

verner_8_combine

`verner_8_combine` — batch 34 builtin.

```
my $result = verner_8_combine $x
# or in a pipeline:
@list |> map verner_8_combine |> p
```

versin

`versin` — trig extensions builtin.

```
my $result = versin $x
# or in a pipeline:
@list |> map versin |> p
```

versine

`versine` — trig extensions builtin. Alias for `versin`.

```
my $result = versine $x
# or in a pipeline:
@list |> map versine |> p
```

version_cmp

`version_cmp` — file stat / path builtin. Alias for `compare_versions`.

```
my $result = version_cmp $x
# or in a pipeline:
@list |> map version_cmp |> p
```

vertex_connectivity

`vertex_connectivity` — batch 75 builtin.

```
my $result = vertex_connectivity $x
# or in a pipeline:
@list |> map vertex_connectivity |> p
```

vertex_cover_lp_round

`vertex_cover_lp_round` — batch 41 builtin.

```
my $result = vertex_cover_lp_round $x
# or in a pipeline:
@list |> map vertex_cover_lp_round |> p
```

vf2_consistent

`vf2_consistent` — batch 75 builtin.

```
my $result = vf2_consistent $x
# or in a pipeline:
@list |> map vf2_consistent |> p
```

vickrey_auction

`vickrey_auction` — batch 73 builtin.

```
my $result = vickrey_auction $x
# or in a pipeline:
@list |> map vickrey_auction |> p
```

vigenere_decrypt

`vigenere_decrypt` — batches 17-18 builtin.

```
my $result = vigenere_decrypt $x
# or in a pipeline:
@list |> map vigenere_decrypt |> p
```

vigenere_encrypt

`vigenere_encrypt` — batches 17-18 builtin.

```
my $result = vigenere_encrypt $x
# or in a pipeline:
@list |> map vigenere_encrypt |> p
```

vin_check

`vin_check` — batch 82 builtin.

```
my $result = vin_check $x
# or in a pipeline:
@list |> map vin_check |> p
```

vincenty_dist

`vincenty_dist` — batch 58 builtin.

```
my $result = vincenty_dist $x
# or in a pipeline:
@list |> map vincenty_dist |> p
```

virtual_temperature_full

`virtual_temperature_full` — batch 42 builtin.

```
my $result = virtual_temperature_full $x
# or in a pipeline:
@list |> map virtual_temperature_full |> p
```

viterbi_pos_tag

`viterbi_pos_tag` — batch 69 builtin.

```
my $result = viterbi_pos_tag $x
# or in a pipeline:
@list |> map viterbi_pos_tag |> p
```

vlookup

`vlookup` — batch 57 builtin.

```
my $result = vlookup $x
# or in a pipeline:
@list |> map vlookup |> p
```

vo2_max_estimate

`vo2_max_estimate` — batch 12 builtin.

```
my $result = vo2_max_estimate $x
# or in a pipeline:
@list |> map vo2_max_estimate |> p
```

void

`void` — conversion / utility (batch 4) builtin.

```
my $result = void $x
# or in a pipeline:
@list |> map void |> p
```

voigt_profile

`voigt_profile` — batch 72 builtin.

```
my $result = voigt_profile $x
# or in a pipeline:
@list |> map voigt_profile |> p
```

voigt_profile_simple

`voigt_profile_simple` — batches 17-18 builtin.

```
my $result = voigt_profile_simple $x
# or in a pipeline:
@list |> map voigt_profile_simple |> p
```

voltage_divider

`voltage_divider` — batch 12 builtin.

```
my $result = voltage_divider $x
# or in a pipeline:
@list |> map voltage_divider |> p
```

voltaic_efficiency

`voltaic_efficiency` — batch 38 builtin.

```
my $result = voltaic_efficiency $x
# or in a pipeline:
@list |> map voltaic_efficiency |> p
```

volume_form_riemannian

`volume_form_riemannian` — batch 39 builtin.

```
my $result = volume_form_riemannian $x
# or in a pipeline:
@list |> map volume_form_riemannian |> p
```

vqe_step

`vqe_step` — batch 80 builtin.

```
my $result = vqe_step $x
# or in a pipeline:
@list |> map vqe_step |> p
```

vsepr_geometry

`vsepr_geometry` — batch 63 builtin.

```
my $result = vsepr_geometry $x
# or in a pipeline:
@list |> map vsepr_geometry |> p
```

w_state

`w_state` — batch 80 builtin.

```
my $result = w_state $x
# or in a pipeline:
@list |> map w_state |> p
```

w_state_n

`w_state_n` — batches 14-16 builtin.

```
my $result = w_state_n $x
# or in a pipeline:
@list |> map w_state_n |> p
```

wacc

`wacc($equity, $debt, $cost_equity, $cost_debt, $tax_rate)` — computes Weighted Average Cost of Capital. Returns the blended cost of capital accounting for the tax shield on debt.

```
p wacc(60, 40, 0.10, 0.05, 0.25)   # ~7.5%
```

wald_test_stat

`wald_test_stat` — batch 36 builtin.

```
my $result = wald_test_stat $x
# or in a pipeline:
@list |> map wald_test_stat |> p
```

walden_product

`walden_product` — batch 38 builtin.

```
my $result = walden_product $x
# or in a pipeline:
@list |> map walden_product |> p
```

walker_circulation_step

`walker_circulation_step` — batch 42 builtin.

```
my $result = walker_circulation_step $x
# or in a pipeline:
@list |> map walker_circulation_step |> p
```

walksat_step

`walksat_step` — batch 67 builtin.

```
my $result = walksat_step $x
# or in a pipeline:
@list |> map walksat_step |> p
```

war_above_replacement

war_above_replacement — batch 56 builtin.

```
my $result = war_above_replacement $x
# or in a pipeline:
@list |> map war_above_replacement |> p
```

warburg_impedance

warburg_impedance — batch 38 builtin.

```
my $result = warburg_impedance $x
# or in a pipeline:
@list |> map warburg_impedance |> p
```

warped_product_step_zero

warped_product_step_zero — batch 39 builtin.

```
my $result = warped_product_step_zero $x
# or in a pipeline:
@list |> map warped_product_step_zero |> p
```

warshall_step

warshall_step — batch 75 builtin.

```
my $result = warshall_step $x
# or in a pipeline:
@list |> map warshall_step |> p
```

wasserstein_dist_emp

wasserstein_dist_emp — batch 40 builtin.

```
my $result = wasserstein_dist_emp $x
# or in a pipeline:
@list |> map wasserstein_dist_emp |> p
```

watershed_segment

watershed_segment — batch 77 builtin.

```
my $result = watershed_segment $x
# or in a pipeline:
@list |> map watershed_segment |> p
```

wattersons_theta

wattersons_theta — batch 33 builtin.

```
my $result = wattersons_theta $x
# or in a pipeline:
@list |> map wattersons_theta |> p
```

watts_to_hp

watts_to_hp — file stat / path builtin.

```
my $result = watts_to_hp $input
```

wave_string_speed

wave_string_speed — batch 23 builtin.

```
my $result = wave_string_speed $x
# or in a pipeline:
@list |> map wave_string_speed |> p
```

wavelength_freq

wavelength_freq — physics formulas builtin.

```
my $result = wavelength_freq $x
# or in a pipeline:
@list |> map wavelength_freq |> p
```

wc

wc — trivial string ops builtin. Alias for word_count.

```
my $result = wc $x
# or in a pipeline:
@list |> map wc |> p
```

wcag_contrast

wcag_contrast — batch 9 builtin. Alias for contrast_ratio_wcag.

```
my $result = wcag_contrast $x
# or in a pipeline:
@list |> map wcag_contrast |> p
```

wcag_luminance

wcag_luminance — batch 9 builtin. Alias for relative_luminance.

```
my $result = wcag_luminance $x
# or in a pipeline:
@list |> map wcag_luminance |> p
```

wcag_pass

wcag_pass — batch 82 builtin.

```
my $result = wcag_pass $x
# or in a pipeline:
@list |> map wcag_pass |> p
```

wcount

wcount — extended stdlib batch 3 builtin. Alias for count_words.

```
my $result = wcount $x
# or in a pipeline:
@list |> map wcount |> p
```

weak_composition_count

weak_composition_count — batch 9 builtin.

```
my $result = weak_composition_count $x
# or in a pipeline:
@list |> map weak_composition_count |> p
```

weaken

`weaken` — list / aggregate builtin.

```
my $result = weaken $x
# or in a pipeline:
@list |> map weaken |> p
```

weber_number

`weber_number` — batch 13 builtin.

```
my $result = weber_number $x
# or in a pipeline:
@list |> map weber_number |> p
```

weber_number_step

`weber_number_step` — batch 42 builtin.

```
my $result = weber_number_step $x
# or in a pipeline:
@list |> map weber_number_step |> p
```

week_number_iso

`week_number_iso` — batch 82 builtin.

```
my $result = week_number_iso $x
# or in a pipeline:
@list |> map week_number_iso |> p
```

week_of_year

`week_of_year` — extended stdlib batch 3 builtin.

```
my $result = week_of_year $x
# or in a pipeline:
@list |> map week_of_year |> p
```

weekday_name

`weekday_name` — date helpers builtin.

```
my $result = weekday_name $x
# or in a pipeline:
@list |> map weekday_name |> p
```

weekday_name_short

`weekday_name_short` — batch 52 builtin.

```
my $result = weekday_name_short $x
# or in a pipeline:
@list |> map weekday_name_short |> p
```

weekday_short

`weekday_short` — date helpers builtin.

```
my $result = weekday_short $x
# or in a pipeline:
@list |> map weekday_short |> p
```

weeks_between

`weeks_between` — batch 9 builtin.

```
my $result = weeks_between $x
# or in a pipeline:
@list |> map weeks_between |> p
```

weibpdf

`weibull_pdf` (alias `weibpdf`) evaluates the Weibull distribution PDF at x with shape k and scale lambda.

```
p weibpdf(1.0, 1.5, 1.0)
```

weighted_match_kuhn_step

`weighted_match_kuhn_step` — batch 41 builtin.

```
my $result = weighted_match_kuhn_step $x
# or in a pipeline:
@list |> map weighted_match_kuhn_step |> p
```

weighted_mean

`weighted_mean` — math / numeric (uncategorized batch) builtin.

```
my $result = weighted_mean $x
# or in a pipeline:
@list |> map weighted_mean |> p
```

weighted_sample

`weighted_sample` — extended stdlib builtin.

```
my $result = weighted_sample $x
# or in a pipeline:
@list |> map weighted_sample |> p
```

weighted_set_cover_step

`weighted_set_cover_step` — batch 41 builtin.

```
my $result = weighted_set_cover_step $x
# or in a pipeline:
@list |> map weighted_set_cover_step |> p
```

weighted_var

`weighted_var` — weighted variance. Args: values, weights.

```
p wvar([1,2,3,4], [1,1,1,1])  # same as var
```

weingarten_map_step

`weingarten_map_step` — batch 39 builtin.

```
my $result = weingarten_map_step $x
# or in a pipeline:
@list |> map weingarten_map_step |> p
```

welch_psd_avg

`welch_psd_avg` — batch 74 builtin.

```
my $result = welch_psd_avg $x
# or in a pipeline:
@list |> map welch_psd_avg |> p
```

welch_ttest

`welch_ttest` (alias `welcht`) performs Welch's t-test for two independent samples with unequal variances. Returns [t-statistic, degrees of freedom].

```
my ($t, $df) = @{welcht([1,2,3,4,5], [3,4,5,6,7])}
p "t=$t df=$df"
```

welch_window

`welch_window` — batch 25 builtin.

```
my $result = welch_window $x
# or in a pipeline:
@list |> map welch_window |> p
```

welsh_powell

`welsh_powell` — batch 75 builtin.

```
my $result = welsh_powell $x
# or in a pipeline:
@list |> map welsh_powell |> p
```

werckmeister_iii

`werckmeister_iii` — batch 64 builtin.

```
my $result = werckmeister_iii $x
# or in a pipeline:
@list |> map werckmeister_iii |> p
```

westerlies_jet_speed

`westerlies_jet_speed` — batch 42 builtin.

```
my $result = westerlies_jet_speed $x
# or in a pipeline:
@list |> map westerlies_jet_speed |> p
```

wet_bulb_potential

`wet_bulb_potential` — batch 42 builtin.

```
my $result = wet_bulb_potential $x
# or in a pipeline:
@list |> map wet_bulb_potential |> p
```

weyl_group_order

`weyl_group_order` — batch 37 builtin.

```
my $result = weyl_group_order $x
# or in a pipeline:
@list |> map weyl_group_order |> p
```

weyl_tensor_step_zero

`weyl_tensor_step_zero` — batch 39 builtin.

```
my $result = weyl_tensor_step_zero $x
# or in a pipeline:
@list |> map weyl_tensor_step_zero |> p
```

wfrm

`wfrm` — go/general functional utilities builtin. Alias for `words_from`.

```
my $result = wfrm $x
# or in a pipeline:
@list |> map wfrm |> p
```

wh

`wh` — filesystem extensions builtin. Alias for `which`.

```
my $result = wh $x
# or in a pipeline:
@list |> map wh |> p
```

wha

Return a list of all absolute paths matching the given command name across every directory in `PATH`, not just the first match. The short alias `wha` keeps things concise. This is useful for detecting shadowed executables or auditing which versions of a tool are installed.

```
my @all = which_all("python3")
@all |> e p                # /usr/local/bin/python3
                             # /usr/bin/python3
p scalar wha("perl")        # number of perls on PATH
```

when_false

`when_false` — functional combinators builtin.

```
my $result = when_false $x
# or in a pipeline:
@list |> map when_false |> p
```

when_true

`when_true` — functional combinators builtin.

```
my $result = when_true $x
# or in a pipeline:
@list |> map when_true |> p
```

which_fn

`which_fn` — return indices where a predicate is true. Like R's which().

```
my @idx = @{which_fn([1,5,3,8,2], fn { $_[0] > 3 })}  # [1, 3]
```

white

`white` — color / ansi builtin. Alias for `red`.

```
my $result = white $x
# or in a pipeline:
@list |> map white |> p
```

white_bold

`white_bold` — color / ansi builtin. Alias for `red`.

```
my $result = white_bold $x
# or in a pipeline:
@list |> map white_bold |> p
```

white_robust_se

`white_robust_se` — batch 36 builtin.

```
my $result = white_robust_se $x
# or in a pipeline:
@list |> map white_robust_se |> p
```

white_test_stat

`white_test_stat` — batch 36 builtin.

```
my $result = white_test_stat $x
# or in a pipeline:
@list |> map white_test_stat |> p
```

whittaker_m

`whittaker_m` — batch 29 builtin.

```
my $result = whittaker_m $x
# or in a pipeline:
@list |> map whittaker_m |> p
```

whole_life_a

`whole_life_a` — batch 60 builtin.

```
my $result = whole_life_a $x
# or in a pipeline:
@list |> map whole_life_a |> p
```

wieferich_check

`wieferich_check` — batch 35 builtin.

```
my $result = wieferich_check $x
# or in a pipeline:
@list |> map wieferich_check |> p
```

wien

`wien` — physics formulas builtin. Alias for `wien_displacement`.

```
my $result = wien $x
# or in a pipeline:
@list |> map wien |> p
```

wien_constant

`wien_constant` — physics constants builtin.

```
my $result = wien_constant $x
# or in a pipeline:
@list |> map wien_constant |> p
```

wien_displacement

`wien_displacement` — physics formulas builtin.

```
my $result = wien_displacement $x
# or in a pipeline:
@list |> map wien_displacement |> p
```

wiener_filter

`wiener_filter` — batch 74 builtin.

```
my $result = wiener_filter $x
# or in a pipeline:
@list |> map wiener_filter |> p
```

wigner_3j_bound

`wigner_3j_bound` — batch 32 builtin.

```
my $result = wigner_3j_bound $x
# or in a pipeline:
@list |> map wigner_3j_bound |> p
```

wilcoxon

`wilcoxon` — batch 53 builtin.

```
my $result = wilcoxon $x
# or in a pipeline:
@list |> map wilcoxon |> p
```

wilson_test

`wilson_test` — batch 35 builtin.

```
my $result = wilson_test $x
# or in a pipeline:
@list |> map wilson_test |> p
```

wilson_theta_step

`wilson_theta_step` — batch 34 builtin.

```
my $result = wilson_theta_step $x
# or in a pipeline:
@list |> map wilson_theta_step |> p
```

win_probability

`win_probability` — batch 56 builtin.

```
my $result = win_probability $x
# or in a pipeline:
@list |> map win_probability |> p
```

wincirc

`wincirc` — additional missing stdlib functions builtin. Alias for `windowed_circular`.

```
my $result = wincirc $x
# or in a pipeline:
@list |> map wincirc |> p
```

wind_chill

`wind_chill` — physics formulas builtin.

```
my $result = wind_chill $x
# or in a pipeline:
@list |> map wind_chill |> p
```

wind_chill_celsius

`wind_chill_celsius` — batch 12 builtin.

```
my $result = wind_chill_celsius $x
# or in a pipeline:
@list |> map wind_chill_celsius |> p
```

window_n

`window_n` — go/general functional utilities builtin.

```
my $result = window_n $x
# or in a pipeline:
@list |> map window_n |> p
```

windowed_circular

`windowed_circular` — additional missing stdlib functions builtin.

```
my $result = windowed_circular $x
# or in a pipeline:
@list |> map windowed_circular |> p
```

winn

`winn` — go/general functional utilities builtin. Alias for `window_n`.

```
my $result = winn $x
# or in a pipeline:
@list |> map winn |> p
```

winsor

`winsor` — math / numeric (uncategorized batch) builtin. Alias for `winsorize`.

```
my $result = winsor $x
# or in a pipeline:
@list |> map winsor |> p
```

winsorize

`winsorize` — math / numeric (uncategorized batch) builtin.

```
my $result = winsorize $x
# or in a pipeline:
@list |> map winsorize |> p
```

winsorized_variance_arr

`winsorized_variance_arr` — batches 14-16 builtin.

```
my $result = winsorized_variance_arr $x
# or in a pipeline:
@list |> map winsorized_variance_arr |> p
```

wire_resistance

`wire_resistance` — batch 12 builtin.

```
my $result = wire_resistance $x
# or in a pipeline:
@list |> map wire_resistance |> p
```

without

`without` — file stat / path builtin.

```
my $result = without $x
# or in a pipeline:
@list |> map without |> p
```

without_nth

`without_nth` — file stat / path builtin.

```
my $result = without_nth $x
# or in a pipeline:
@list |> map without_nth |> p
```

wmean

`wmean` — math / numeric (uncategorized batch) builtin. Alias for `weighted_mean`.

```
my $result = wmean $x
# or in a pipeline:
@list |> map wmean |> p
```

woba_weight

`woba_weight` — batch 56 builtin.

```
my $result = woba_weight $x
# or in a pipeline:
@list |> map woba_weight |> p
```

wofz

`wofz` — batch 72 builtin.

```
my $result = wofz $x
# or in a pipeline:
@list |> map wofz |> p
```

wolfe_strong_q

`wolfe_strong_q` — batch 19 builtin.

```
my $result = wolfe_strong_q $x
# or in a pipeline:
@list |> map wolfe_strong_q |> p
```

woodall_number

`woodall_number` — batch 11 builtin.

```
my $result = woodall_number $x
# or in a pipeline:
@list |> map woodall_number |> p
```

word_alignment_ibm1

`word_alignment_ibm1` — batch 69 builtin.

```
my $result = word_alignment_ibm1 $x
# or in a pipeline:
@list |> map word_alignment_ibm1 |> p
```

word_alignment_ibm2

`word_alignment_ibm2` — batch 69 builtin.

```
my $result = word_alignment_ibm2 $x
# or in a pipeline:
@list |> map word_alignment_ibm2 |> p
```

word_count

`word_count` — trivial string ops builtin.

```
my $result = word_count $x
# or in a pipeline:
@list |> map word_count |> p
```

word_count_simple

`word_count_simple` — batch 11 builtin. Alias for `string_words_count`.

```
my $result = word_count_simple $x
# or in a pipeline:
@list |> map word_count_simple |> p
```

word_frequencies

`word_frequencies` — string processing (uncategorized batch) builtin.

```
my $result = word_frequencies $x
# or in a pipeline:
@list |> map word_frequencies |> p
```

word_similarity

`word_similarity` — batch 70 builtin.

```
my $result = word_similarity $x
# or in a pipeline:
@list |> map word_similarity |> p
```

word_wrap_text

`word_wrap_text` — extended stdlib builtin.

```
my $result = word_wrap_text $x
# or in a pipeline:
@list |> map word_wrap_text |> p
```

wordfreq

`wordfreq` — string processing (uncategorized batch) builtin. Alias for `word_frequencies`.

```
my $result = wordfreq $x
# or in a pipeline:
@list |> map wordfreq |> p
```

words_from

`words_from` — go/general functional utilities builtin.

```
my $result = words_from $x
# or in a pipeline:
@list |> map words_from |> p
```

work

`work($force, $distance, $angle)` — compute work W = F×d×cos(θ) (Joules). Angle in degrees, default 0.

```
p work(100, 10)       # 1000 J (force parallel to motion)
p work(100, 10, 60)   # 500 J (force at 60°)
```

woy

`woy` — extended stdlib batch 3 builtin. Alias for `week_of_year`.

```
my $result = woy $x
# or in a pipeline:
@list |> map woy |> p
```

wpa

`wpa` — batch 56 builtin.

```
my $result = wpa $x
# or in a pipeline:
@list |> map wpa |> p
```

wrap_around

`wrap_around` — trig / math (batch 2) builtin.

```
my $result = wrap_around $x
# or in a pipeline:
@list |> map wrap_around |> p
```

wrap_index

`wrap_index` — list helpers (batch 4) builtin.

```
my $result = wrap_index $x
# or in a pipeline:
@list |> map wrap_index |> p
```

wrap_range

`wrap_range` — extended stdlib builtin.

```
my $result = wrap_range $x
# or in a pipeline:
@list |> map wrap_range |> p
```

wrap_text

`wrap_text` — extended stdlib builtin.

```
my $result = wrap_text $x
# or in a pipeline:
@list |> map wrap_text |> p
```

wrc_plus

`wrc_plus` — batch 56 builtin.

```
my $result = wrc_plus $x
# or in a pipeline:
@list |> map wrc_plus |> p
```

wright_omega

`wright_omega` — batch 29 builtin.

```
my $result = wright_omega $x
# or in a pipeline:
@list |> map wright_omega |> p
```

writhe_polygon

`writhe_polygon` — batch 37 builtin.

```
my $result = writhe_polygon $x
# or in a pipeline:
@list |> map writhe_polygon |> p
```

wrprng

`wrprng` — extended stdlib builtin. Alias for `wrap_range`.

```
my $result = wrprng $x
# or in a pipeline:
@list |> map wrprng |> p
```

wrpt

`wrpt` — extended stdlib builtin. Alias for `wrap_text`.

```
my $result = wrpt $x
# or in a pipeline:
@list |> map wrpt |> p
```

wsamp

`wsamp` — extended stdlib builtin. Alias for `weighted_sample`.

```
my $result = wsamp $x
# or in a pipeline:
@list |> map wsamp |> p
```

wwrap

`wwrap` — extended stdlib builtin. Alias for `word_wrap_text`.

```
my $result = wwrap $x
# or in a pipeline:
@list |> map wwrap |> p
```

www_auth_parse

`www_auth_parse` — batch 81 builtin.

```
my $result = www_auth_parse $x
# or in a pipeline:
@list |> map www_auth_parse |> p
```

wyhash_mix

`wyhash_mix` — batch 26 builtin.

```
my $result = wyhash_mix $x
# or in a pipeline:
@list |> map wyhash_mix |> p
```

wynn_epsilon

`wynn_epsilon` — batches 14-16 builtin.

```
my $result = wynn_epsilon $x
# or in a pipeline:
@list |> map wynn_epsilon |> p
```

x25519_field_mul

`x25519_field_mul` — batches 17-18 builtin.

```
my $result = x25519_field_mul $x
# or in a pipeline:
@list |> map x25519_field_mul |> p
```

xadd

`xadd` — batch 71 builtin.

```
my $result = xadd $x
# or in a pipeline:
@list |> map xadd |> p
```

xd

`xd` — serialization (stryke-only encoders) builtin. Alias for `xml_decode`.

```
my $result = xd $x
# or in a pipeline:
@list |> map xd |> p
```

xe

`xe` — serialization (stryke-only encoders) builtin. Alias for `xml_encode`.

```
my $result = xe $x
# or in a pipeline:
@list |> map xe |> p
```

xentropy

`xentropy` — math / numeric (uncategorized batch) builtin. Alias for `cross_entropy`.

```
my $result = xentropy $x
# or in a pipeline:
@list |> map xentropy |> p
```

xfip

`xfip` — batch 56 builtin.

```
my $result = xfip $x
# or in a pipeline:
@list |> map xfip |> p
```

xirr

`xirr(\@cashflows, \@dates)` — computes IRR for irregularly-spaced cash flows. Dates can be epoch timestamps or date strings. More accurate than `irr` for real-world investments.

```
my @cf = (-1000, 500, 600)
my @dates = ('2020-01-01', '2020-07-01', '2021-01-01')
p xirr(\@cf, \@dates)
```

xlen

`xlen` — batch 71 builtin.

```
my $result = xlen $x
# or in a pipeline:
@list |> map xlen |> p
```

xlookup

`xlookup` — batch 57 builtin.

```
my $result = xlookup $x
# or in a pipeline:
@list |> map xlookup |> p
```

xmlagg

xmlagg — batch 70 builtin.

```
my $result = xmlagg $x
# or in a pipeline:
@list |> map xmlagg |> p
```

xmlforest

xmlforest — batch 70 builtin.

```
my $result = xmlforest $x
# or in a pipeline:
@list |> map xmlforest |> p
```

xnpv

xnpv — batch 57 builtin.

```
my $result = xnpv $x
# or in a pipeline:
@list |> map xnpv |> p
```

xor_bool

xor_bool — boolean combinators builtin.

```
my $result = xor_bool $x
# or in a pipeline:
@list |> map xor_bool |> p
```

xor_break_singlebyte

xor_break_singlebyte — batch 35 builtin.

```
my $result = xor_break_singlebyte $x
# or in a pipeline:
@list |> map xor_break_singlebyte |> p
```

xor_brute_keylen

xor_brute_keylen — batches 17-18 builtin.

```
my $result = xor_brute_keylen $x
# or in a pipeline:
@list |> map xor_brute_keylen |> p
```

xor_byte_string

xor_byte_string — batches 17-18 builtin.

```
my $result = xor_byte_string $x
# or in a pipeline:
@list |> map xor_byte_string |> p
```

xor_cipher_byte

`xor_cipher_byte` — batch 26 builtin.

```
my $result = xor_cipher_byte $x
# or in a pipeline:
@list |> map xor_cipher_byte |> p
```

xor_strings

`xor_strings` — string helpers (batch 4) builtin.

```
my $result = xor_strings $x
# or in a pipeline:
@list |> map xor_strings |> p
```

xorshift32

`xorshift32` — batch 26 builtin.

```
my $result = xorshift32 $x
# or in a pipeline:
@list |> map xorshift32 |> p
```

xorshift32_step

`xorshift32_step` — batch 9 builtin.

```
my $result = xorshift32_step $x
# or in a pipeline:
@list |> map xorshift32_step |> p
```

xorshift64

`xorshift64` — batch 26 builtin.

```
my $result = xorshift64 $x
# or in a pipeline:
@list |> map xorshift64 |> p
```

xrange

`xrange` — batch 71 builtin.

```
my $result = xrange $x
# or in a pipeline:
@list |> map xrange |> p
```

xtea_round

`xtea_round` — batches 14-16 builtin.

```
my $result = xtea_round $x
# or in a pipeline:
@list |> map xtea_round |> p
```

xyz_to_lab

`xyz_to_lab` — batch 30 builtin.

```
my $result = xyz_to_lab $input
```

xyz_to_rgb

`xyz_to_rgb` — batch 82 builtin.

```
my $result = xyz_to_rgb $input
```

yamanouchi_word_count

`yamanouchi_word_count` — batch 37 builtin.

```
my $result = yamanouchi_word_count $x
# or in a pipeline:
@list |> map yamanouchi_word_count |> p
```

yards_per_attempt

`yards_per_attempt` — batch 56 builtin.

```
my $result = yards_per_attempt $x
# or in a pipeline:
@list |> map yards_per_attempt |> p
```

yards_to_m

`yards_to_m` — unit conversions builtin.

```
my $result = yards_to_m $input
```

yd

`yd` — serialization (stryke-only encoders) builtin. Alias for `yaml_decode`.

```
my $result = yd $x
# or in a pipeline:
@list |> map yd |> p
```

ye

`ye` — serialization (stryke-only encoders) builtin. Alias for `yaml_encode`.

```
my $result = ye $x
# or in a pipeline:
@list |> map ye |> p
```

years_between

`years_between` — batch 9 builtin.

```
my $result = years_between $x
# or in a pipeline:
@list |> map years_between |> p
```

yellow

`yellow` — color / ansi builtin. Alias for `red`.

```
my $result = yellow $x
# or in a pipeline:
@list |> map yellow |> p
```

yellow_bold

`yellow_bold` — color / ansi builtin. Alias for `red`.

```
my $result = yellow_bold $x
# or in a pipeline:
@list |> map yellow_bold |> p
```

yen_k_shortest

`yen_k_shortest` — batch 75 builtin.

```
my $result = yen_k_shortest $x
# or in a pipeline:
@list |> map yen_k_shortest |> p
```

yesterday

`yesterday` — date (batch 2) builtin.

```
my $result = yesterday $x
# or in a pipeline:
@list |> map yesterday |> p
```

yield_to_maturity

`yield_to_maturity` — batch 57 builtin.

```
my $result = yield_to_maturity $input
```

yielddisc

`yielddisc` — batch 81 builtin.

```
my $result = yielddisc $x
# or in a pipeline:
@list |> map yielddisc |> p
```

yieldmat

`yieldmat` — batch 81 builtin.

```
my $result = yieldmat $x
# or in a pipeline:
@list |> map yieldmat |> p
```

ylll_pml

`ylll_pml` — batch 61 builtin.

```
my $result = ylll_pml $x
# or in a pipeline:
@list |> map ylll_pml |> p
```

yn

`yn` — batch 72 builtin.

```
my $result = yn $x
# or in a pipeline:
@list |> map yn |> p
```

ytm_newton

`ytm_newton` — batch 20 builtin.

```
my $result = ytm_newton $x
# or in a pipeline:
@list |> map ytm_newton |> p
```

yuv_to_rgb

`yuv_to_rgb` — batch 82 builtin.

```
my $result = yuv_to_rgb $input
```

z_array

`z_array` — batch 10 builtin.

```
my $result = z_array $x
# or in a pipeline:
@list |> map z_array |> p
```

z_func

`z_func` — batch 10 builtin. Alias for `z_array`.

```
my $result = z_func $x
# or in a pipeline:
@list |> map z_func |> p
```

z_function_alt

`z_function_alt` — batches 14-16 builtin.

```
my $result = z_function_alt $x
# or in a pipeline:
@list |> map z_function_alt |> p
```

z_score

`z_score($value, $mean, $stddev)` — computes the z-score (standard score) of a value given the population mean and standard deviation. Returns how many standard deviations the value is from the mean. Useful for standardizing data and detecting outliers.

```
my $z = z_score(85, 75, 10)
p $z   # 1 (one stddev above mean)
p z_score(50, 60, 5)   # -2
```

z_score_robust

`z_score_robust` — batch 11 builtin.

```
my $result = z_score_robust $x
# or in a pipeline:
@list |> map z_score_robust |> p
```

z_scores

`z_scores(@data)` — computes z-scores for all values in a list, returning an arrayref of standardized values. Each z-score indicates how many standard deviations that value is from the mean of the dataset. Useful for comparing values across different scales.

```
my @grades = (70, 80, 90)
my $zs = z_scores(@grades)
p @$zs   # (-1, 0, 1)
@data |> z_scores |> e p
```

zadd

`zadd` — batch 71 builtin.

```
my $result = zadd $x
# or in a pipeline:
@list |> map zadd |> p
```

zalgo

`zalgo` — string processing (uncategorized batch) builtin.

```
my $result = zalgo $x
# or in a pipeline:
@list |> map zalgo |> p
```

zall

`zall` — go/general functional utilities builtin. Alias for `zip_all`.

```
my $result = zall $x
# or in a pipeline:
@list |> map zall |> p
```

zcard

`zcard` — batch 71 builtin.

```
my $result = zcard $x
# or in a pipeline:
@list |> map zcard |> p
```

zcount

`zcount` — batch 71 builtin.

```
my $result = zcount $x
# or in a pipeline:
@list |> map zcount |> p
```

zcross

`zcross` — matrix operations (uncategorized batch) builtin. Alias for `zero_crossings`.

```
my $result = zcross $x
# or in a pipeline:
@list |> map zcross |> p
```

zeeman_energy

`zeeman_energy` — batch 32 builtin.

```
my $result = zeeman_energy $x
# or in a pipeline:
@list |> map zeeman_energy |> p
```

zeller_day_of_week

`zeller_day_of_week` — batch 13 builtin.

```
my $result = zeller_day_of_week $x
# or in a pipeline:
@list |> map zeller_day_of_week |> p
```

zeno_survival

`zeno_survival` — batch 32 builtin.

```
my $result = zeno_survival $x
# or in a pipeline:
@list |> map zeno_survival |> p
```

zero

`zero` — trivial numeric / predicate builtins builtin.

```
my $result = zero $x
# or in a pipeline:
@list |> map zero |> p
```

zero_crossings

`zero_crossings` — matrix operations (uncategorized batch) builtin.

```
my $result = zero_crossings $x
# or in a pipeline:
@list |> map zero_crossings |> p
```

zero_order_concentration

`zero_order_concentration` — batch 21 builtin.

```
my $result = zero_order_concentration $x
# or in a pipeline:
@list |> map zero_order_concentration |> p
```

zero_pad

`zero_pad` — batch 25 builtin.

```
my $result = zero_pad $x
# or in a pipeline:
@list |> map zero_pad |> p
```

zero_sum_minmax

`zero_sum_minmax` — batch 43 builtin.

```
my $result = zero_sum_minmax $x
# or in a pipeline:
@list |> map zero_sum_minmax |> p
```

zeros

`zeros` — matrix / linear algebra builtin.

```
my $result = zeros $x
# or in a pipeline:
@list |> map zeros |> p
```

zeros_mat

`zeros_mat` — matrix / linear algebra builtin. Alias for `zeros_matrix`.

```
my $result = zeros_mat $x
# or in a pipeline:
@list |> map zeros_mat |> p
```

zeros_matrix

`zeros_matrix` — matrix / linear algebra builtin.

```
my $result = zeros_matrix $x
# or in a pipeline:
@list |> map zeros_matrix |> p
```

zeta_potential_estimate

`zeta_potential_estimate` — batch 38 builtin.

```
my $result = zeta_potential_estimate $x
# or in a pipeline:
@list |> map zeta_potential_estimate |> p
```

zfill

`zfill` — extended stdlib builtin.

```
my $result = zfill $x
# or in a pipeline:
@list |> map zfill |> p
```

zfill_num

`zfill_num` — extended stdlib builtin.

```
my $result = zfill_num $x
# or in a pipeline:
@list |> map zfill_num |> p
```

zgemm

`zgemm` — batch 66 builtin.

```
my $result = zgemm $x
# or in a pipeline:
@list |> map zgemm |> p
```

zincrby

`zincrby` — batch 71 builtin.

```
my $result = zincrby $x
# or in a pipeline:
@list |> map zincrby |> p
```

zip_all

`zip_all` — go/general functional utilities builtin.

```
my $result = zip_all $x
# or in a pipeline:
@list |> map zip_all |> p
```

zip_apply

`zip_apply` — functional combinators builtin.

```
my $result = zip_apply $x
# or in a pipeline:
@list |> map zip_apply |> p
```

zip_archive

`zip_archive` — crypto / encoding builtin. Alias for `zip_create`.

```
my $result = zip_archive $x
# or in a pipeline:
@list |> map zip_archive |> p
```

zip_central_dir

`zip_central_dir` — batch 62 builtin.

```
my $result = zip_central_dir $x
# or in a pipeline:
@list |> map zip_central_dir |> p
```

zip_eocd

`zip_eocd` — batch 62 builtin.

```
my $result = zip_eocd $x
# or in a pipeline:
@list |> map zip_eocd |> p
```

zip_fill

`zip_fill` — python/ruby stdlib builtin.

```
my $result = zip_fill $x
# or in a pipeline:
@list |> map zip_fill |> p
```

zip_local_header

`zip_local_header` — batch 62 builtin.

```
my $result = zip_local_header $x
# or in a pipeline:
@list |> map zip_local_header |> p
```

zip_with

`zip_with` — functional / iterator builtin. Alias for `take_while`.

```
my $result = zip_with $x
# or in a pipeline:
@list |> map zip_with |> p
```

zipf

`zipf` — python/ruby stdlib builtin. Alias for `zip_fill`.

```
my $result = zipf $x
# or in a pipeline:
@list |> map zipf |> p
```

zipl

`zipl` — python/ruby stdlib builtin.

```
my $result = zipl $x
# or in a pipeline:
@list |> map zipl |> p
```

zipmap

`zipmap` — file stat / path builtin.

```
my $result = zipmap $x
# or in a pipeline:
@list |> map zipmap |> p
```

ziv_lempel_estimate

`ziv_lempel_estimate` — batch 40 builtin.

```
my $result = ziv_lempel_estimate $x
# or in a pipeline:
@list |> map ziv_lempel_estimate |> p
```

zlexcount

`zlexcount` — batch 71 builtin.

```
my $result = zlexcount $x
# or in a pipeline:
@list |> map zlexcount |> p
```

zpk2sos_step

`zpk2sos_step` — batch 74 builtin.

```
my $result = zpk2sos_step $x
# or in a pipeline:
@list |> map zpk2sos_step |> p
```

zpk2tf_step

`zpk2tf_step` — batch 74 builtin.

```
my $result = zpk2tf_step $x
# or in a pipeline:
@list |> map zpk2tf_step |> p
```

zrangebyscore

`zrangebyscore` — batch 71 builtin.

```
my $result = zrangebyscore $x
# or in a pipeline:
@list |> map zrangebyscore |> p
```

zrank

`zrank` — batch 71 builtin.

```
my $result = zrank $x
# or in a pipeline:
@list |> map zrank |> p
```

zrem

`zrem` — batch 71 builtin.

```
my $result = zrem $x
# or in a pipeline:
@list |> map zrem |> p
```

zrevrank

`zrevrank` — batch 71 builtin.

```
my $result = zrevrank $x
# or in a pipeline:
@list |> map zrevrank |> p
```

zrlc

`zrlc` — physics formulas builtin. Alias for `impedance_rlc`.

```
my $result = zrlc $x
# or in a pipeline:
@list |> map zrlc |> p
```

zscore

`zscore` — file stat / path builtin.

```
my $result = zscore $x
# or in a pipeline:
@list |> map zscore |> p
```

zscore_count

`zscore_count` — batch 33 builtin.

```
my $result = zscore_count $x
# or in a pipeline:
@list |> map zscore_count |> p
```

zscore_norm

`zscore_norm` — batch 27 builtin.

```
my $result = zscore_norm $x
# or in a pipeline:
@list |> map zscore_norm |> p
```

zscores

`zscores` — statistics (extended) builtin. Alias for `z_scores`.

```
my $result = zscores $x
# or in a pipeline:
@list |> map zscores |> p
```

zst

`zst` — crypto / encoding builtin. Alias for `zstd`.

```
my $result = zst $x
# or in a pipeline:
@list |> map zst |> p
```

zstd_encode_step

`zstd_encode_step` — batch 81 builtin.

```
my $result = zstd_encode_step $x
# or in a pipeline:
@list |> map zstd_encode_step |> p
```

zstd_frame_header

`zstd_frame_header` — batch 62 builtin.

```
my $result = zstd_frame_header $x
# or in a pipeline:
@list |> map zstd_frame_header |> p
```

zstd_window_size_log

`zstd_window_size_log` — batch 40 builtin.

```
my $result = zstd_window_size_log $x
# or in a pipeline:
@list |> map zstd_window_size_log |> p
```