Ruby Cheat Sheet

Ruby 4.0 (4.0.7) · checked 2026-10-18 · Irurueta Docs · AI-assisted; verify against docs.ruby-lang.org/en/4.0

TOOLS & FLAGS

`ruby f.rb` run · `irb` REPL · `ri Array#map` docs · `gem` · `bundle` · `rake -T` · `rdbg f.rb` · `rbs validate` · `rubocop -a`

```
ruby -e 'puts 1+1'   -w warnings   -c
syntax check
ruby -n/-p -l -a -i  one-liners   -r
lib   -I dir
ruby --yjit f.rb     --zjit
(experimental)
ruby -W:no-experimental   RUBYOPT=-w
```

GEMFILE & GEMSPEC

```
source "https://rubygems.org"
gemspec
gem "rake", "~> 13.0"
group :development, :test do
  gem "minitest"; gem "rubocop"
end
# shelf.gemspec
Gem::Specification.new do |s|
  s.name = "shelf"; s.version =
"0.1.0"
  s.summary = "Library catalogue"
  s.license = "MIT"
  s.required_ruby_version = ">= 4.0"
  s.bindir = "exe"; s.executables =
["shelf"]
  s.add_dependency "json", "~> 2.9"
  s.metadata["rubygems_mfa_required"]
= "true"
end
```

`bundle init|install|add x|update|exec|outdated` · `bundle gem shelf` · `gem build|push` · `Gemfile.lock` committed for apps. **Default** gems ship with Ruby and are locked in; **bundled** gems (benchmark, logger, ostruct…) need a Gemfile line when bundling.

FILE BASICS

```
# frozen_string_literal: true   #
first lines
require "json"            # gem /
stdlib
require_relative "shelf/book"
autoload :Loan, "shelf/loan"
__FILE__ == $PROGRAM_NAME   # run as
script
```

`snake_case` methods/vars · `CamelCase` classes · `SCREAMING` consts · `?` predicate · `!` dangerous · `_` unused · 2-space indent.

LITERALS

```
1_000  0xff  0b1  0o7  1.5e3  3r  2i
"a#{1+1}"  'raw'  :sym  :"a b"  %w[a
b]  %i[a b]
<<~TXT           # squiggly heredoc
  indented
TXT
[1, 2]  {a: 1, "b" => 2}  {x:, y:}  #
y: shorthand
(1..3) (1...3) (1..) (..5)  /re/i
%r{a/b}
-> (a, b = 2) { a + b }   # call: l.
(1) l[1]
```

Falsy: only `nil` and `false` (0, "", [] are truthy). `p({a: 1, "b" => 2})` prints `{a: 1, "b" => 2}` (3.4+).

VARIABLES & ASSIGNMENT

`x`	local (block/method scope)
`@x`	instance
`@@x`	class var (avoid; use class-level `@x`)
`$x`	global
`X`	constant (reassign warns)

```
a ||= 1   a &&= 2   a += 1
a, b = b, a
first, *rest = [1, 2, 3]
a, (b, *c), d = 1, [2, 3, 4], 5
h => {name:}        # rightward
destructure
x = if c then 1 else 2 end   #
expression
```

OPERATORS & PRECEDENCE

High → low: `!` `~` `+@` > `**` > `-@` > `*` `/` `%` > `+` `-` > `<<` `>>` > `&` > `|` `^` > `<` `<=` `>` `>=` > `<=>` `==` `===` `=~` `!=` > `&&` > `||` > `..` `...` > `?:` > modifier-rescue > `=` `op=` > `not` > `and` `or` > modifier-if/unless/while.

`&&`/`||` bind tighter than `and`/`or` (control flow only). `obj&.m` nil-safe · `===` case equality · `=~` regexp · `<=>` returns -1/0/1/nil · `7 / 2 #=> 3`, `-7 / 2 #=> -4`, `7.fdiv(2) #=> 3.5`.

CONTROL

```
if a then b elsif c then d else e end
x = 1 if ok   puts "no" unless ok
case v
when 1..5, Integer then :int
when /re/ then :re
else :other end
while c; end  until c; end  loop {
break }
3.times { }  1.upto(3) { }  for i in
1..3 do end
break  next  redo  return
catch(:done) { throw :done, 42 }   #=>
42
```

METHODS & ARGUMENTS

```
def m(a, b = 1, *rest, c, k:, k2: 2,
      **opts, &blk) = ...   # this
order
def area = w * h              #
endless
def fwd(...) = target(...)
def anon(*, **, &) = target(*, **, &)
m(1, *arr, k: 1, **h, &:sym)
def a? = true   def x=(v); @x = v; end
```

Keyword args are separate from positional hashes (3.0). Defaults evaluate per call. Implicit return = last expression. `private` / `protected` / `public`; `private def f`; `private_class_method`; implicit-receiver call or literal `self.f`.

BLOCKS, PROCS, LAMBDAS

```
[1, 2].map { it * 2 }      # 3.4 it
[1, 2].map { _1 + 1 }      # numbered
param
def each = block_given? ? yield(1) :
to_enum
%w[a b].map(&:upcase)  m =
1.method(:+)
f = :upcase.to_proc >>
:reverse.to_proc
f.call("ab")   #=> "BA"
```

	proc	lambda
args	lenient, arity 2	strict
return	returns from method	returns from lambda

EXCEPTIONS

```
begin
  risky
rescue KeyError, TypeError => e
  warn e.message
else   ok
ensure  cleanup
end    # retry re-runs begin
class AppError < StandardError; end
raise AppError, "msg"   raise X.new,
cause: e
```

Bare `rescue` = `StandardError`. Never rescue `Exception`. `Exception` → `NoMemoryError`, `ScriptError` (`LoadError`, `NotImplementedError`, `SyntaxError`), `SecurityError`, `SignalException` (`Interrupt`), `SystemExit`, `SystemStackError`, `StandardError` (`ArgumentError`, `IOError`, `IndexError`→`KeyError`/`StopIteration`, `NameError`→`NoMethodError`, `RangeError`, `RuntimeError`→`FrozenError`, `TypeError`, `ZeroDivisionError`, `NoMatchingPatternError`).

OBJECT MODEL

```
class Book < Item
  include Comparable   # after class
in ancestors
  prepend Audit        # before class
  extend Finders       # class-level
methods
  attr_reader :title; attr_accessor :n
  def initialize(title) = @title =
title
  class << self; def build = new("t");
end
  def <=>(o) = title <=> o.title
  def to_s = "Book #{title}"
end
Book.ancestors #=> [Audit, Book,
Comparable, Item, ...]
```

Lookup: singleton → prepended → class → included (last first) → superclass. `super` (same args) · `super()` (none) · `super(a)`. Refinements: `refine String do…end`; `using M` (file-scoped). `module_function`.

EQUALITY & COPYING

`equal?` identity · `==` value · `eql?`+`hash` Hash key (`1.eql?(1.0) #=> false`) · `===` case. Define `==`, `eql?`, `hash` together; `include Comparable` adds `< > between? clamp`.

```
a.dup      # shallow, unfrozen
a.clone    # shallow, keeps frozen &
singleton
a.clone(freeze: false)  a.freeze
a.frozen?
FrozenError: can't modify frozen
Array: [1]
```

CORE TYPES

```
7.divmod(2) #=> [3, 1]  10.pow(3, 7)
#=> 6
Integer.sqrt(17) #=> 4  1r/3 #=> (1/3)
2**-1 #=> (1/2)
"abc"[1..] #=> "bc"  "a-b".tr("-","_")
"%05.1f|%-4s|%+d" % [3.14159, "ab", 5]
"x".sub(/x/) { it.upcase }  s.freeze
+"" / -""
:ab.to_proc  "ab".to_sym  :ab.name
[1,2,3].first(2)  [1,[2]].flatten
a[1..]  a.dig(0,1)
h.fetch(:k)  #=> KeyError "key not
found: :k"
h.dig(:a, 0, :b)  h.transform_values {
it + 1 }
Set[1, 2] | Set[3] #=> Set[1, 2, 3]
# core in 4.0
(1..10).step(3).to_a  (1..).lazy.map {
}.first(3)
Time.utc(2026,10,18).iso8601
Time.now.strftime("%F")
Process.clock_gettime(
  Process::CLOCK_MONOTONIC)
```

STRUCT vs DATA

```
Book = Data.define(:isbn, :title) do
  def label = "#{title} (#{isbn})"
end
b = Book.new(isbn: "1", title: "T")
b.with(title: "U")  b.to_h  #
immutable
#=> #<data Book isbn="1", title="T">
Loan = Struct.new(:a, :b,
keyword_init: true)
#=> #<struct Loan a=1, b=2>  # mutable
```

Data: immutable value objects, keyword or positional, `with`, pattern-matchable. Struct: mutable, enumerable.

REGEXP

```
m = "2026-10".match(/(?<y>\d+)-(?
<m>\d+)/)
m[:y] #=> "2026"   $~  $1
/(?<yr>\d+)/ =~ s   # makes local yr
s.scan(/\d+/)  s.gsub(/a/, "b")  s =~
/re/
\A \z anchors (not ^ $)
Regexp.timeout = 1.0
```

ENUMERABLE

```
map select reject filter_map flat_map
find
reduce(:+)  sum  min_by  max_by
sort_by { -it }
group_by(&:odd?)  partition  tally
zip
each_slice(2)  each_cons(2)
each_with_index
each_with_object({})  take_while
chunk_while
%w[a b a].tally #=> {"a" => 2, "b" =>
1}
(1..6).each_slice(2).to_a #=> [[1, 2],
[3, 4], [5, 6]]
```

Include `Enumerable` + define `each`. No block → `Enumerator`: `next`, `peek`, `with_index`, `lazy`, `Enumerator.new { |y| y << 1 }`.

PATTERN MATCHING

```
case cfg
in {name: String => n, roles: [:a,
*rest]} then ..
in [Integer => x, *] | [] then ..
in [*, String => s, *post] then ..   #
find
in Integer | Float => num if num > 0
then ..
in ^v | ^(v + 1) then ..
# pin
in nil then ..
end
42 in Integer     #=> true   (boolean)
{a: 1} => {a:}    # binds a, raises
NoMatchingPatternError
```

Custom: define `deconstruct` (array) / `deconstruct_keys(keys)` (hash).

METAPROGRAMMING

```
obj.send(:x)   obj.public_send(:x)  #
respects private
Foo.define_method(:dbl) { n * 2 }
def method_missing(name, *args)
  name.start_with?("get_") ?
name.to_s.delete_prefix("get_") :
super
end
def respond_to_missing?(n, priv =
false) = n.start_with?("get_") ||
super
instance_eval { @a }  class_eval { }
def self.inherited(k) = (super; ...)
# hooks:
included extended prepended
method_added const_added
obj.instance_variables
Foo.instance_methods(false)
```

Never `send`/`eval`/`constantize` untrusted input; use allow-lists.

CONCURRENCY

```
t = Thread.new { 5 }; t.value #=> 5
t.join
m = Mutex.new; m.synchronize { }
q = Queue.new; q << 1; q.pop
SizedQueue.new(10)
f = Fiber.new { Fiber.yield 1; 2 }
f.resume #=> 1   f.resume #=> 2
Fiber.set_scheduler(s)  # non-blocking
I/O
# Ractor (experimental, 4.0 API)
p = Ractor::Port.new
Ractor.new(p) { |pt| pt << 42 };
p.receive #=> 42
Ractor.new { 1 + 1 }.value #=> 2
```

GVL: one thread runs Ruby code at a time; threads overlap on I/O. CPU parallelism → processes or Ractors. Set `report_on_exception`. `Ractor.yield`/`take` are gone in 4.0.

STDLIB ONE-LINERS

```
File.read(p)  File.write(p, s)
File.open(p) { |f| }
Dir.glob("*.rb")
Pathname("a/b.rb").extname
Open3.capture2("echo", "hi") #=>
["hi\n", status]
system("ls", "-l")   # array form, no
shell
Net::HTTP.get(URI("https://x.test/a"))
JSON.parse(s, symbolize_names: true)
h.to_json
YAML.safe_load(s, symbolize_names:
true)
OptionParser.new { |o| o.on("-v") { }
}.parse!
Logger.new($stdout).info("x")
CSV.read(p, headers: true)
Benchmark.realtime { work }
GC.stat(:count)
```

Never `Marshal.load`/`YAML.unsafe_load` untrusted data.

TESTING & QUALITY

```
require "minitest/autorun"
class BookTest < Minitest::Test
  def test_title = assert_equal "T",
Book.new("T").title
end
describe Book do   # Minitest::Spec
  it("has a title") {
_(Book.new("T").title).must_equal "T"
}
end
RSpec.describe Book do
  let(:b) { Book.new("T") }
  it { expect(b.title).to eq("T") }
  let(:repo) { instance_double(Repo,
find: nil) }
end
# sig/shelf.rbs
class Shelf::Book
  attr_reader title: String
  def initialize: (String title) ->
void
end
rubocop -a   rubocop -A   rubocop --
auto-gen-config
```

PERFORMANCE & VM

`--yjit` or `RUBY_YJIT_ENABLE=1` or `RubyVM::YJIT.enable` · `--zjit` experimental · Prism parser → YARV bytecode · MJIT/RJIT gone · `GC.stat`, `GC.start` · `Benchmark.realtime { }`, benchmark-ips (community) · `Process.clock_gettime` with `CLOCK_MONOTONIC` for timing. Generational + incremental GC; tune with `RUBY_GC_*` env vars.

C extensions: `extconf.rb` + `mkmf`; Fiddle (bundled gem 4.0); `ffi` gem; ruby.wasm for browsers/WASI.

CHANGED SINCE OLDER TUTORIALS (with version)

3.0 keyword args separated from positional hashes
3.0–3.2 `$SAFE` / taint removed
3.2 `Fixnum` / `Bignum`, `File.exists?` removed
3.2 Struct hacks → `Data.define`
3.4 `_1` → `it`
3.4 chilled string literals warn on mutation (`-W:deprecated`)
3.4 `{:a=>1}` → `{a: 1}` inspect; new error format `undefined method 'x' for an instance of Foo`
3.4–4.0 `OpenStruct`, `benchmark`, `logger`… become bundled gems (Gemfile)
4.0 `require "set"` not needed: core `Set`; `Pathname` core
4.0 `Kernel#open("|cmd")` removed (use `IO.popen`/`Open3`)
4.0 `Ractor.yield`/`take` → `Ractor::Port`, `#value`
3.3/4.0 MJIT (3.3) and RJIT (4.0) removed: YJIT (and experimental ZJIT)