

Ruby Quick Reference

Objects, blocks, iterators, regex, file I/O essentials

Basics

Hello World

```
puts "Hello, World!"
print "no newLine"
p [1, 2, 3] # inspect output: [1, 2, 3]
```

Run Ruby

```
ruby script.rb # run a file
ruby -e 'puts "hi"' # run inline
irb # interactive REPL
```

Variables

name	Local variable
@name	Instance variable
@@count	Class variable
\$debug	Global variable
MAX_SIZE	Constant (uppercase by convention)

Types

42.class	# Integer
3.14.class	# Float
"hello".class	# String
true.class	# TrueClass
nil.class	# NilClass
:symbol.class	# Symbol

Strings

String Basics

```
name = "World"
puts "Hello, #{name}!" # interpolation (double quotes)
puts 'No #{interpolation}' # literal (single quotes)
multi = <<-HEREDOC
  indented heredoc
HEREDOC
```

String Methods

.length / .size	Character count
.upcase / .downcase	Case conversion
.strip	Remove leading/trailing whitespace
.split(' ,')	Split into array
.gsub(/pat/, 'rep')	Global substitution
.include?('sub')	Check if contains substring
.start_with?('pre')	Check prefix
.chars / .bytes	Array of characters / bytes
.to_i / .to_f	Convert to integer / float
.freeze	Make string immutable

Arrays & Hashes

Arrays

```
arr = [1, "two", :three]
arr << 4 # push (append)
arr[0] # 1
arr[-1] # 4 (last element)
arr[1..2] # ["two", :three] (slice)
```

Array Methods

.push / .pop	Add/remove from end
.shift / .unshift	Remove/add from beginning
.flatten	Flatten nested arrays
.compact	Remove nil values
.uniq	Remove duplicates
.sort / .reverse	Sort / reverse order
.map { x x * 2 }	Transform each element
.select { x x > 0 }	Filter elements
.reduce(0) { sum, x sum + x }	Accumulate into single value

Hashes

```
user = { name: "Alice", age: 30 } # symbol keys
old = { "key" => "value" } # string keys
user[:name] # "Alice"
user[:email] = "a@b.com" # add pair
user.fetch(:name, "default") # with default
```

Hash Methods

.keys / .values	Array of keys / values
.each { k, v }	Iterate key-value pairs
.merge(other)	Merge two hashes
.key?(k) / .value?(v)	Check existence
.select { k, v }	Filter pairs
.transform_values { v }	Transform all values

Control Flow

Conditionals

```
if score >= 90 then "A"
elsif score >= 80 then "B"
else "C"
end
puts "adult" if age >= 18 # inline if
puts "minor" unless age >= 18 # inline unless
```

Case / When

```
case status
when :ok then puts "success"
when :error then puts "failed"
when 400..499 then puts "client error"
else puts "unknown"
end
```

Loops

```
5.times { |i| puts i }
(1..10).each { |n| puts n }
while condition do end
until condition do end
loop { break if done }
```

Ternary & Logical

```
status = age >= 18 ? "adult" : "minor"
name = input || "default" # or-assign
name ||= "fallback" # same effect
```

Methods

Defining Methods

```
def greet(name, greeting = "Hello")
  "#{greeting}, #{name}!"
end
greet("Alice") # "Hello, Alice!"
greet("Bob", "Hi") # "Hi, Bob!"
```

Return Values

```
def add(a, b)
  a + b # last expression is implicit return
end
def divide(a, b)
  return nil if b == 0
  a.to_f / b
end
```

Keyword & Splat Arguments

```
def connect(host:, port: 80, **opts)
  puts "#{host}:#{port} #{opts}"
end
def log(*messages)
  messages.each { |m| puts m }
end
```

Method Conventions

method?	Returns boolean (predicate)
method!	Mutates receiver (bang method)
self.method	Class method definition

Classes

Class Definition

```
class User
  attr_accessor :name, :email
  def initialize(name, email)
    @name = name
    @email = email
  end
end
```

Inheritance

```
class Admin < User
  def initialize(name, email, level)
    super(name, email)
    @level = level
  end
end
```

Access Control

public	Default; accessible from anywhere
private	Only accessible within the class
protected	Accessible within class and subclasses
attr_reader	Generate getter method
attr_writer	Generate setter method
attr_accessor	Generate getter and setter

Modules

Mixins

```
module Greetable
  def greet
    "Hello, I'm #{name}"
  end
end
class User; include Greetable; end
```

Namespaces

```
module Payment
  class Processor
    def charge(amount) end
  end
end
p = Payment::Processor.new
```

Ruby Quick Reference

Include vs Extend

include ModName	Add as instance methods
extend ModName	Add as class methods
prepend ModName	Insert before class in method lookup

Blocks & Iterators

Block Syntax

```
[1, 2, 3].each { |n| puts n }      # single-line block
[1, 2, 3].each do |n|
  puts n                          # multi-line block
end
```

Yield

```
def with_logging
  puts "start"
  result = yield
  puts "end"
  result
end
with_logging { expensive_operation }
```

Procs & Lambdas

```
square = Proc.new { |x| x ** 2 }
square.call(5)          # 25
double = ->(x) { x * 2 } # lambda
double.call(3)         # 6
[1, 2, 3].map(&square)  # [1, 4, 9]
```

Common Iterators

.each	Iterate over elements
.map / .collect	Transform each element
.select / .filter	Keep matching elements
.reject	Remove matching elements
.reduce / .inject	Accumulate into single value
.each_with_index	Iterate with index
.flat_map	Map and flatten one level
.any? / .all? / .none?	Boolean checks on collection

Regex

Matching

```
"hello 42" =~ /\d+/      # 6 (match position)
"hello" =~ /\d+/        # nil (no match)
"hello".match?(/ell/)   # true
md = "age: 30".match(/(\d+)/)
md[1]                   # "30"
```

Common Patterns

/^start/	Anchored at beginning
/end\$/	Anchored at end
/\d+/	One or more digits
/\w+/	Word characters
/\s+/	Whitespace
/[a-z]+/i	Case-insensitive
/(group)/	Capture group

Substitution

```
"hello world".sub(/world/, "Ruby") # first match
"aabba".gsub(/a/, "x")            # all matches: "xxbbx"
"foo bar".gsub(/(\w+)/) { $1.upcase } # "FOO BAR"
```

File I/O

Read & Write

```
content = File.read("data.txt")
lines = File.readlines("data.txt", chomp: true)
File.write("out.txt", "hello\n")
File.open("log.txt", "a") { |f| f.puts "entry" }
```

File Operations

File.exist?(path)	Check if file exists
File.directory?(path)	Check if path is a directory
File.basename(path)	Filename without directory
File.extname(path)	File extension
File.size(path)	File size in bytes
File.delete(path)	Delete a file
Dir.glob('*.*rb')	Find files matching pattern
FileUtils.mkdir_p(path)	Create directory recursively

CSV & JSON

```
require "json"
data = JSON.parse(File.read("data.json"))
File.write("out.json", JSON.pretty_generate(data))
require "csv"
CSV.foreach("data.csv", headers: true) { |row| puts row["name"] }
```