

Referência Rápida de Ruby

Objetos, blocos, iteradores, regex, E/S de arquivos

Básico

Hello World

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

Executar Ruby

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

Variáveis

name	Variável local
@name	Variável de instância
@@count	Variável de classe
\$debug	Variável global
MAX_SIZE	Constante (maiúsculas por convenção)

Tipos

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

Strings

Noções Básicas de Strings

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

Métodos de String

.length / .size	Contagem de caracteres
.upcase / .downcase	Conversão de capitalização
.strip	Remover espaços no início/fim
.split(' ,')	Dividir em array
.gsub(/pat/, 'rep')	Substituição global
.include?('sub')	Verificar se contém substring
.start_with?('pre')	Verificar prefixo
.chars / .bytes	Array de caracteres / bytes
.to_i / .to_f	Converter para inteiro / float
.freeze	Tornar string imutável

Arrays e Hashes

Arrays

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

Métodos de Array

.push / .pop	Adicionar/remover do final
.shift / .unshift	Remover/adicionar do início
.flatten	Achatar arrays aninhados
.compact	Remover valores nil
.uniq	Remover duplicatas
.sort / .reverse	Ordenar / inverter ordem
.map { x x * 2 }	Transformar cada elemento
.select { x x > 0 }	Filtrar elementos
.reduce(0) { sum, x sum + x }	Acumular em valor único

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
```

Métodos de Hash

.keys / .values	Array de chaves / valores
.each { k, v }	Iterar pares chave-valor
.merge(other)	Mesclar dois hashes
.key?(k) / .value?(v)	Verificar existência
.select { k, v }	Filtrar pares
.transform_values { v }	Transformar todos os valores

Controle de Fluxo

Condicionais

```
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
```

Laços

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

Ternário e Lógico

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

Métodos

Definindo Métodos

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

Valores de Retorno

```
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
```

Argumentos Keyword e Splat

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

Convenções de Método

method?	Retorna booleano (predicado)
method!	Muta o receptor (método bang)
self.method	Definição de método de classe

Classes

Definição de Classe

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

Herança

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

Controle de Acesso

public	Padrão; acessível de qualquer lugar
private	Apenas acessível dentro da classe
protected	Acessível na classe e subclasses
attr_reader	Gerar método getter
attr_writer	Gerar método setter
attr_accessor	Gerar getter e setter

Módulos

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
```

Referência Rápida de Ruby

Include vs Extend

include ModName	Adicionar como métodos de instância
extend ModName	Adicionar como métodos de classe
prepend ModName	Inserir antes da classe na busca de métodos

Blocos e Iteradores

Sintaxe de Bloco

```
[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 e 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]
```

Iteradores Comuns

.each	Iterar sobre elementos
.map / .collect	Transformar cada elemento
.select / .filter	Manter elementos correspondentes
.reject	Remover elementos correspondentes
.reduce / .inject	Acumular em valor único
.each_with_index	Iterar com índice
.flat_map	Mapear e achatar um nível
.any? / .all? / .none?	Verificações booleanas na coleção

Regex

Correspondência

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

Padrões Comuns

/^start/	Ancorado no início
/end\$/	Ancorado no fim
/\d+/	Um ou mais dígitos
/\w+/	Caracteres de palavra
/\s+/	Espaço em branco
/[a-z]+/i	Insensível a maiúsculas
/(group)/	Grupo de captura

Substituição

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

E/S de Arquivos

Ler e Escrever

```
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" }
```

Operações com Arquivos

File.exist?(path)	Verificar se arquivo existe
File.directory?(path)	Verificar se é diretório
File.basename(path)	Nome do arquivo sem diretório
File.extname(path)	Extensão do arquivo
File.size(path)	Tamanho do arquivo em bytes
File.delete(path)	Excluir um arquivo
Dir.glob('*.*rb')	Encontrar arquivos por padrão
FileUtils.mkdir_p(path)	Criar diretório recursivamente

CSV e 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"] }
```