

RUBY クイックリファレンス

オブジェクト、ブロック、イテレータ、正規表現、ファイル I/O の基本

基本

Hello World

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

Ruby の実行

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

変数

name ローカル変数
@name インスタンス変数
@count クラス変数
\$debug グローバル変数
MAX_SIZE 定数 (慣例として大文字)

型

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

文字列

文字列の基本

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

文字列メソッド

.length / .size 文字数
.upcase / .downcase 大文字/小文字変換
.strip 前後の空白を削除
.split(' ', ' ') 配列に分割
.gsub(/pat/, 'rep') グローバル置換
.include?('sub') 部分文字列を含むか確認
.start_with?('pre') プレフィックスを確認
.chars / .bytes 文字/バイトの配列
.to_i / .to_f 整数/浮動小数点数に変換
.freeze 文字列をイミュータブルにする

配列とハッシュ

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

配列メソッド

.push / .pop 末尾に追加/末尾から削除
.shift / .unshift 先頭から削除/先頭に追加
.flatten ネストした配列をフラット化
.compact nil 値を削除
.uniq 重複を削除
.sort / .reverse ソート/逆順
.map { |x| x * 2 } 各要素を変換
.select { |x| x > 0 } 要素をフィルタリング
.reduce(0) { |sum, x| sum + x } 単一値に集約

ハッシュ

```
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
```

ハッシュメソッド

.keys / .values キー/値の配列
.each { |k, v| } キーと値のペアを反復
.merge(others) 2つのハッシュをマージ
.key?(k) / .value?(v) 存在確認
.select { |k, v| } ペアをフィルタリング
.transform_values { |v| } 全値を変換

制御フロー

条件文

```
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
```

ループ

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

三項演算子と論理演算子

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

メソッド

メソッドの定義

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

戻り値

```
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
```

キーワード引数とスプラット引数

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

メソッドの命名規則

method? 真偽値を返す (述語)
method! レシーバーを変更する (破壊的メソッド)
self.method クラスメソッドの定義

クラス

クラスの定義

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

継承

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

アクセス制御

public デフォルト: どこからでもアクセス可能
private クラス内からのみアクセス可能
protected クラスとサブクラスからアクセス可能
attr_reader ゲッターメソッドを生成
attr_writer セッターメソッドを生成
attr_accessor ゲッターとセッターを生成

モジュール

ミックスイン

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

名前空間

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

Include vs Extend

include ModName インスタンスメソッドとして追加
extend ModName クラスメソッドとして追加
prepend ModName メソッド探索のクラス前に挿入

ブロックとイテレータ

ブロックの構文

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

Yield

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

Proc と Lambda

```
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]
```

よく使うイテレータ

.each 要素を反復
.map / .collect 各要素を変換
.select / .filter 一致する要素を保持
.reject 一致する要素を削除
.reduce / .inject 単一値に集約
.each_with_index インデックス付きで反復
.flat_map mapして1レベルフラット化

.any? / .all? / .none? コレクションの真偽チェック

正規表現

マッチング

```
"hello 42" =~ /\d+/ # 6 (match position)
"hello" =~ /\d+/ # nil (no match)
"hello".match?(/\ell/) # true
m0 = "age: 30".match(/(\d+)/)
m0[1] # "30"
```

よくあるパターン

/^start/ 先頭でアンカー
/end\$/ 末尾でアンカー
/\d+/ 1以上の数字
/\w+/ 単語文字
/\s+/ 空白
/[a-z]+/i 大文字小文字を区別しない
/(group)/ キャプチャグループ

置換

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

ファイル I/O

読み書き

```
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" }
```

ファイル操作

File.exist?(path) ファイルが存在するか確認
File.directory?(path) バスがディレクトリか確認
File.basename(path) ディレクトリなしのファイル名
File.extname(path) ファイル拡張子
File.size(path) バイト単位のファイルサイズ
File.delete(path) ファイルを削除
Dir.glob(*.rb) パターンに一致するファイルを検索
FileUtils.mkdir_p(path) ディレクトリを再帰的に作成

CSV と 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"] }
```