RÉFÉRENCE RAPIDE RUBY

Objets, blocs, itérateurs, regex, E/S fichier essentiels

Bases

Hello World

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

Exécuter Ruby

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

Variables

`name`	Variable locale
`@name`	Variable d'instance
`@@count`	Variable de classe
`$debug`	Variable globale
`MAX_SIZE`	Constante (majuscules par convention)

Types

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

Chaînes

Bases des chaînes

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

Méthodes de chaîne

`.length / .size`	Nombre de caractères
`.upcase / .downcase`	Conversion de casse
`.strip`	Supprimer les espaces de début/fin
`.split(',')`	Diviser en tableau
`.gsub(/pat/, 'rep')`	Substitution globale
`.include?('sub')`	Vérifier si contient une sous-chaîne
`.start_with?('pre')`	Vérifier le préfixe
`.chars / .bytes`	Tableau de caractères / octets
`.to_i / .to_f`	Convertir en entier / flottant
`.freeze`	Rendre la chaîne immuable

Tableaux et hashes

Tableaux

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

Méthodes de tableau

`.push / .pop`	Ajouter/supprimer à la fin
`.shift / .unshift`	Supprimer/ajouter au début
`.flatten`	Aplatir les tableaux imbriqués
`.compact`	Supprimer les valeurs nil
`.uniq`	Supprimer les doublons
`.sort / .reverse`	Trier / inverser l'ordre
`.map { |x| x * 2 }`	Transformer chaque élément
`.select { |x| x > 0 }`	Filtrer les éléments
`.reduce(0) { |sum, x| sum + x }`	Accumuler en une seule valeur

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
```

Méthodes de hash

`.keys / .values`	Tableau de clés / valeurs
`.each { |k, v| }`	Itérer les paires clé-valeur
`.merge(other)`	Fusionner deux hashes
`.key?(k) / .value?(v)`	Vérifier l'existence
`.select { |k, v| }`	Filtrer les paires
`.transform_values { |v| }`	Transformer toutes les valeurs

Flux de contrôle

Conditionnelles

```
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
```

Boucles

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

Ternaire et logique

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

Méthodes

Définir des méthodes

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

Valeurs de retour

```
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
```

Arguments par mot-clé et splat

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

Conventions de méthode

`method?`	Retourne un booléen (prédicat)
`method!`	Modifie le récepteur (méthode bang)
`self.method`	Définition de méthode de classe

Classes

Définition de classe

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

Héritage

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

Contrôle d'accès

`public`	Par défaut ; accessible de partout
`private`	Accessible uniquement dans la classe
`protected`	Accessible dans la classe et les sous-classes
`attr_reader`	Générer une méthode getter
`attr_writer`	Générer une méthode setter
`attr_accessor`	Générer getter et setter

Modules

Mixins

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

Espaces de noms

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

Include vs Extend

`include ModName`	Ajouter comme méthodes d'instance
`extend ModName`	Ajouter comme méthodes de classe
`prepend ModName`	Insérer avant la classe dans la recherche de méthode

Blocs et itérateurs

Syntaxe des blocs

```
[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 et 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]
```

Itérateurs courants

`.each`	Itérer sur les éléments
`.map / .collect`	Transformer chaque élément
`.select / .filter`	Conserver les éléments correspondants
`.reject`	Supprimer les éléments correspondants
`.reduce / .inject`	Accumuler en une seule valeur
`.each_with_index`	Itérer avec index
`.flat_map`	Map et aplatir d'un niveau
`.any? / .all? / .none?`	Vérifications booléennes sur la collection

Regex

Correspondance

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

Motifs courants

`/^start/`	Ancré au début
`/end$/`	Ancré à la fin
`/\d+/`	Un ou plusieurs chiffres
`/\w+/`	Caractères de mot
`/\s+/`	Espace blanc
`/[a-z]+/i`	Insensible à la casse
`/(group)/`	Groupe capturant

Substitution

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

E/S fichier

Lire et écrire

```
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" }
```

Opérations sur les fichiers

`File.exist?(path)`	Vérifier si le fichier existe
`File.directory?(path)`	Vérifier si le chemin est un répertoire
`File.basename(path)`	Nom de fichier sans répertoire
`File.extname(path)`	Extension du fichier
`File.size(path)`	Taille du fichier en octets
`File.delete(path)`	Supprimer un fichier
`Dir.glob('*.rb')`	Trouver les fichiers correspondant au motif
`FileUtils.mkdir_p(path)`	Créer un répertoire récursivement

CSV et 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"] }
```