RUBY 빠른 참조

객체, 블록, 이터레이터, 정규식, 파일 I/O 핵심

기본

Hello World

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

Ruby 실행

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

변수

`name`	지역 변수
`@name`	인스턴스 변수
`@@count`	클래스 변수
`$debug`	전역 변수
`MAX_SIZE`	상수 (관례상 대문자)

타입

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

문자열

문자열 기본

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

문자열 메서드

`.length / .size`	문자 수
`.upcase / .downcase`	대소문자 변환
`.strip`	앞뒤 공백 제거
`.split(',')`	배열로 분리
`.gsub(/pat/, 'rep')`	전역 치환
`.include?('sub')`	부분 문자열 포함 여부
`.start_with?('pre')`	접두사 확인
`.chars / .bytes`	문자 / 바이트 배열
`.to_i / .to_f`	정수 / 실수로 변환
`.freeze`	문자열 불변으로 만들기

배열 및 해시

배열

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

배열 메서드

`.push / .pop`	끝에 추가/제거
`.shift / .unshift`	시작에서 제거/추가
`.flatten`	중첩 배열 평탄화
`.compact`	nil 값 제거
`.uniq`	중복 제거
`.sort / .reverse`	정렬 / 역순
`.map { |x| x * 2 }`	각 원소 변환
`.select { |x| x > 0 }`	원소 필터링
`.reduce(0) { |sum, x| sum + x }`	단일 값으로 누적

해시

```
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
```

해시 메서드

`.keys / .values`	키 / 값 배열
`.each { |k, v| }`	키-값 쌍 반복
`.merge(other)`	두 해시 병합
`.key?(k) / .value?(v)`	존재 여부 확인
`.select { |k, v| }`	쌍 필터링
`.transform_values { |v| }`	모든 값 변환

제어 흐름

조건문

```
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
```

반복문

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

삼항 및 논리

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

메서드

메서드 정의

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

반환값

```
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
```

키워드 및 스플랫 인수

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

메서드 관례

`method?`	불리언 반환 (술어)
`method!`	수신자를 변경 (뱅 메서드)
`self.method`	클래스 메서드 정의

클래스

클래스 정의

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

상속

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

접근 제어

`public`	기본값; 어디서나 접근 가능
`private`	클래스 내부에서만 접근 가능
`protected`	클래스 및 서브클래스에서 접근 가능
`attr_reader`	게터 메서드 생성
`attr_writer`	세터 메서드 생성
`attr_accessor`	게터와 세터 생성

모듈

믹스인

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

네임스페이스

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

Include vs Extend

`include ModName`	인스턴스 메서드로 추가
`extend ModName`	클래스 메서드로 추가
`prepend ModName`	메서드 탐색에서 클래스 앞에 삽입

블록 및 이터레이터

블록 문법

```
[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 }
```

Proc 및 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]
```

일반 이터레이터

`.each`	원소 반복
`.map / .collect`	각 원소 변환
`.select / .filter`	일치하는 원소 유지
`.reject`	일치하는 원소 제거
`.reduce / .inject`	단일 값으로 누적
`.each_with_index`	인덱스와 함께 반복
`.flat_map`	맵 후 한 단계 평탄화
`.any? / .all? / .none?`	컬렉션에 대한 불리언 검사

정규식

매칭

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

일반 패턴

`/^start/`	시작에 고정
`/end$/`	끝에 고정
`/\d+/`	하나 이상의 숫자
`/\w+/`	단어 문자
`/\s+/`	공백
`/[a-z]+/i`	대소문자 무시
`/(group)/`	캡처 그룹

치환

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

파일 I/O

읽기 및 쓰기

```
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" }
```

파일 작업

`File.exist?(path)`	파일 존재 여부 확인
`File.directory?(path)`	경로가 디렉터리인지 확인
`File.basename(path)`	디렉터리 없는 파일 이름
`File.extname(path)`	파일 확장자
`File.size(path)`	파일 크기 (바이트)
`File.delete(path)`	파일 삭제
`Dir.glob('*.rb')`	패턴과 일치하는 파일 찾기
`FileUtils.mkdir_p(path)`	재귀적으로 디렉터리 생성

CSV 및 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"] }
```