Ruby Scripting language

CONCEPTS

Everything is an object and almost every call takes an optional block: 3.times { |i| ... }.

The last expression of a method or block is its return value; no return needed.

RUNNING RUBY

ruby -e '*<code>*'
Execute code from the command line

ruby *<file>*.rb [args]
Run a script (args in ARGV)

ruby -c *<file>*.rb
Syntax check only

ruby -w *<file>*.rb
Enable warnings

ruby -r *<lib>* -e '*<code>*'
Require a library first

ruby -I *<dir>* *<file>*.rb
Add a dir to the load path

ruby -n -e '*<code>*'
Run code for each input line

ruby -p -e '*<code>*'
Like -n, then print each line

ruby -lane 'puts $F[0]'
Split lines into fields in $F

IRB

irb
Start the console

ls *<obj>*
List methods and variables of an object

binding.irb
Breakpoint: open irb at this line

GEMS

gem install *<name>*
Install a gem

gem install *<name>* -v *<version>*
Install a specific version

gem list
List installed gems

gem uninstall *<name>*
Remove a gem

gem update *<name>*
Update a gem

SYNTAX

puts *<x>*
Print with a newline

p *<x>*
Print inspect form, returns the value

name = "#{first} #{last}"
Interpolate into a double-quoted string

def *<name>*(a, b = 1, *rest, k:, **opts)
Define a method

if *<a>* ... elsif ... else ... end
Conditional

<obj>*&.*<method>
Safe navigation: nil if obj is nil

x ||= *<default>*
Assign only if nil or false

case x when 1..5 then ... else ... end
Match with ===

:name %i[a b]
Symbols and symbol array

{ a: 1, "b" => 2 }
Hash literals

&:upcase
Pass a method name as a block

yield / block_given?
Call the method's block

STRINGS & OUTPUT

format("%05.1f %s", 3.14159, "x")
printf-style formatting

str.split(",")
Split into an array

str.gsub(/*<re>*/, "*<new>*")
Replace all matches

COLLECTIONS

arr.each { |x| ... }
Iterate

arr.map { |x| x * 2 }
Transform each element

arr.select { |x| x.even? }
Keep matches (reject inverts)

arr.reduce(0) { |sum, x| sum + x }
Fold into one value

COLLECTIONS (cont.)

arr.sort_by { |x| x.size }
Sort by a key

arr.group_by { |x| x.size }
Hash of key ⇒ elements

arr.each_with_object({}) { |x, h| ... }
Build a hash or array while iterating

h.fetch(:k)
Value or KeyError

h.dig(:a, :b)
Safe nested lookup

h.each { |k, v| ... }
Iterate pairs

PATTERN MATCHING

case x in { name: String => n, age: } ... end
Match and bind a hash

case x in [a, *rest] ... end
Match an array and bind

***<expr>* => { name: }**
Destructure, raise if no match

***<expr>* in { role: :admin }**
Boolean match, no raise

CLASSES, STRUCT & DATA

class *<Name>* ... end
Define a class

attr_reader :a / attr_accessor :a
Generate getters / getters+setters

def initialize(a) ; @a = a ; end
Constructor

Point = Struct.new(:x, :y)
Mutable record with accessors

ERRORS & REQUIRES

begin ... rescue Err => e ... ensure ... end
Handle errors

raise ArgumentError, "*<msg>*"
Raise an exception

require "*<lib>*"
Load a library or gem