Ruby Scripting language

START HERE

`ruby -e '<code>'`
Run a one-liner

`ruby -n -e '<code>'`
Loop over input lines

`ruby <file>.rb`
Run a script

`bundle exec <cmd>`
Run with the Gemfile gems

`irb`
Interactive console

`ruby -c <file>.rb`
Check syntax only

`ruby -w <file>.rb`
Run with warnings

`gem install <name>`
Install a gem

Input line stdin or files → `gets` → **$_** current line → `your code` → **Output** -p prints $_

ruby -n / -p loop

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.

Methods ending in ? return booleans; ending in ! mutate or raise (map! vs map).

RUNNING RUBY

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

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

`ruby -v`
Print version, enable verbose mode

`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

RUNNING RUBY (cont.)

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

`ruby -i.bak -pe '<code>' <file>`
Edit files in place with backup

IRB

`irb`
Start the console

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

`show_doc <Class#method>`
Show documentation inline

`show_source <Class#method>`
Show a method's source

`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 install <name> --user-install`
Install into your home dir

`gem list`
List installed gems

`gem search <regexp>`
Search remote gems

GEMS (cont.)

`gem uninstall <name>`
Remove a gem

`gem outdated`
List gems with newer versions

`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

`x = cond ? a : b`
Ternary

`<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 ===

`3.times { |i| ... }`
Repeat n times

SYNTAX (cont.)

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

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

`->(x) { x + 1 }.call(2)`
Lambda; also .(2) and [2]

`&: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

`str.strip / chomp`
Trim whitespace / trailing newline

COLLECTIONS

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

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

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

COLLECTIONS (cont.)

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

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

`arr.group_by { |x| x.size }`
Hash of key => elements

`arr.each_with_index`
Pairs of element and index

`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

`Point = Data.define(:x, :y)`
Immutable value object

ERRORS & REQUIRES

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

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

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

`require_relative "<path>"`
Load a file relative to this one

`ARGV / ENV["<VAR>"]`
Script args / environment

`File.read("<f>") / File.readlines`
Read a file / lines