Learn to Code with Ruby 2.0

10.8) Monkey Patching II

```
class Fixnum
  def seconds
    self
  end
  def minutes
    self * 60
  end
  def hours
    self * 60 * 60
  end
  def days
    self * 60 * 60 * 24
  end
end

puts Time.now + 45.minutes

puts Time.now + 4.hours
puts Time.now + 10.days
```

→ Yielding to a block.

```
class Fixnum
  def custom_times
    i = 0
    while i < self
      yield(i + 1)   # 1, 2, 3, 4, 5
      i += 1
    end
  end
end

5.times { |i| puts i } → 0 1 2 3 4

5.custom_times { |i| puts i } → 1 2 3 4 5
```

20.9) Monkey Patching III

//

20.10) Class variables & Class methods

- Class Variables
  - Some pieces of data extend beyond the scope of a single object;
  - Solves the issue of data duplication across instances; belongs to the whole class rather than to one object.
  - Store info that does not pertain to an instance
- Class Methods
  - method for Class; eg. '.new'.
- Syntax
  - Class Variables: @@; Instance variables: @;
  - Class methods must be prefixed with 'self' keyword.
    - * In instance method def, 'self' refer to the object.

```
class Bicycle
  @@maker = "Biketech"
  @@count = 0
  def self.description          # or Bicycle.description
    "nenenene pupu"
  end
end
puts Bicycle.description  => nenenene pupu.

def self.count                  puts Bicycle.count => 0
  @@count
end
def maker                       => called directly by its instance.
  @@maker
end
```

```
def initialize
  @@count += 1      -> count += 1 whenever initialized
end
```

Questions:

* protected method: A method that can only be called inside the object or by another object of the same class.
* private method: A method that can only be called inside the object.
* monkey patching: Modifying a class after it has been declared.

21) Classes III

21.1) Inheritance

- A way for a class to obtain all methods from another class; avoids duplicate code across multiple classes;
- superclass/parent class ← (inherit) subclass/child class
- Ruby only supports single inheritance. (only inherit from 1 class)
- subclass is 'a type of' superclass, a subset of the superclass. This is called "is-a" relationship.
  (eg. Car is a Vehicle but a Vehicle is not guaranteed to be a Car.)
- Subclasses are followed by a '<' symbol & the name of superclass

21.2) Create subclasses.

```
class Employee
  attr_accessor :age
  attr_reader :name
  def initialize(name, age)
    @name = name
    @age = age
  end
  def introduce
    "Hi, I'm #{name} and I'm #{age} year old."
  end
end
```

```
class Manager < Employee
end
class Worker < Employee
end

bob = Manager.new("Bob", 48)
dan = Worker.new("Daniel", 36)
p bob.introduce ⇝ "Yo. I'm Bob & I'm 48 year old"
p dan.introduce ⇝ "Yo, I'm Daniel & I'm 36 year old".
```

21.3) Check Inheritance Hierarchy

```
p Manager.ancestors ⇝ [Manager, Employee, Object, ...]
p Manager.superclass ⇝ Employee
puts Manager < Employee ⇝ true.
```

21.4) .is_a? .instance_of? Predicate Methods.

↳ to validate ~~that~~ whether an object inherits from a certain class or is made from certain class.

```
puts bob.is_a?(Manager) ⇝ true ⇝ true if it's ancestors.
puts dan.is_a?(Employee) ⇝ true.
puts dan.is_a?(Worker) ⇝ true.
```

* ~~Is~~ .instance_of? ⇝ true if it's direct superclass

21.5) Exclusive instance methods in Subclasses. ✓

21.6) Override Methods in Subclasses ✓.

~~21.7)~~

21.7) The 'super' Keyword

- used to customize the design of subclasses by keeping some of the functionality from the superclass while still adding on some unique functionality in the subclass.
  - ex: to build on top of superclass methods, rather than rebuilding them, to adhere to DRY.

```
class Manager < Employee
  def initialize(name, age, rank)
    super(name, age)
    @rank = rank
  end
  def introduce
    result = super
    result += " I'm also a manager!"
  end
end
```

21.8) The 'super' keyword #.

- 3 ways
  - without ()
  - with () but no arguments.
  - with() with arguments.

```
class Car
  def drive
    "Room! Room!"
  end
end
class Firetruck < Car
  def drive(speed)
    super() + " Beep! Beep! I'm driving at #{speed} miles per hr."
  end
end
ft = Firetruck.new
p ft.drive(45)
```

case ① without ()

```
def drive
  super + " Beep! Beep!
end
```

p ft.drive → Room! Room! Beep! Beep!

case ②

21.10) Singleton Classes & Singleton Methods

- Singleton Method - method that only exists on a single instance of an object that we instantiate from a class.

```
class Player
  def play_game
    rand(1..100) > 50 ? "Winner" : "Loser!"
  end
end

bob = Player.new
boris = Player.new

def boris.play_game
  "Winner!"
end

p bob.play_game
p boris.play_game
p boris.singleton_methods
p bob.singleton_methods
p boris.singleton_class
```

→ 50% winner/loser

→ Singleton Method.

→ 100% winner

→ "Winner!" / "Loser!"

→ "Winner!"

→ [:play_game]

→ []

→ #<Class:...>

- ~~Used~~ Whenever you have just one object & it needs to be isolated & needs to be granted some kind of exclusive functionality, ~~rather~~ rather than defining a brand new subclass for that object to inherit, we can use this Singleton Method.

21.11) Hash as Initialize Argument I

```
class Candidate
  attr_accessor :name, :age, :occupation, :hobby
  def initialize(name, details)
    @name = name
    @age = details[:age]
    @occupation = details[:occupation]
    @hobby = details[:hobby]
  end
end
```

~~*~~ info = {hobby: "chess", age: "25", occupation: "student"}

~~*~~ senator = Candidate.new("Mr. Kyle", info)

p senator.age → 25 ✓

~~p senator.birthplace → nil (not defined~~

21.12) Hash as initialize Argument 2.

* def initialize(name, details = {}) → default
  ~~default = {age: 35,~~

* senator = Candidate.new("Kyle", {hobby: "chess", age: "25"})
  optional optional

* defaults = {age: 35, occupation: "Candidate", hobby: "Running"}
  defaults.merge!(details)

  ↳ To return valid stuffs rather than nil if key is not given

  @age = defaults[:age]
  @occupation = defaults[:occupation]
  @hobby = defaults[:hobby]

↳ To reduce P(errors) & mistakes.

Summary

* Singleton method : A method that only exists on a single object.
* [1,2,3].is_a?(Object) → true
  [1,2,3].instance_of?(Object) → false
* super without () - call the same method in the superclass & pass all current method arguments to it
  without arguments
  super with () - Call the same method in the superclass & pass no arguments to it.
  without arguments
* ~~Ext~~ Subclass < Superclass

22) Reading & Writing Documentation

22.1) www.rubydoc.info/stdlib/core

22.2) Class Method Documentation on RubyDocs. //

22.3) Install rdoc

└ gem install rdoc

22.4) Use rdoc to Generate Your Own Documentation

```
rdoc
 └ album.rb
    └
    ┌ class Album
        include Enumerable
        attr_reader :songs
        def initialize
          @songs = []
        end
        def add_song(song)
          songs << song
        end
        def each
          songs.each do |song|
            yield song
          end
        end
      end
```

Terminal > rdoc

└ rdoc album.rb

// Files : 1

⋮

Classes : 1 (-1 undocumented)

⋮ ⋮

↳ doc folder generated !!

└ index.html → (open)

↳

* click to toggle source ...

22.4) rdoc II. + Comments.

```
# An Album class that ...
class ...
# ...
attr ...
# ...
def ...
```

rdoc album.rb

↳

100% documented.

↳ index → Album → ...

1) Getting Started

- pwd : print working directory.
- RVM : Ruby Version Manager
- Ruby is installed by default in MacOs (But often out-dated)
  - ruby -v
  - rvm.io
- bash profile file : a file that tells our terminal who to look for commands.
  - open ~/.bash_profile
    - source ~/.rvm/scripts/rvm → look at here whenever we type 'rvm' on terminal.
    - source ~/.bash_profile
- rvm install 2.__ → latest version of Ruby.
- clear
- rvm use 2.__ → version eg. 2.3.3
- rvm --default use 2.3.3 → set it as default.
- rvm docs generate-ri → download Ruby doc
- ruby -v ✓

→ Install Atom Text Editor.

- atom.io
- Atom > Preferences > Install > Install Package > Install (Packages) atom-runner → setting to look for its spec
  - > monokai-light → (themes)
  - > Editor > Tab Length : 2
  - > Themes > Atom-Light (UI Theme) ; (Syntax Theme) Monokai-light

→ New file.

- ~~ctrl+n~~ cmd+N
- ~~ctrl+s~~ cmd+S → .rb

→ Run Ruby File from Terminal
- go to the directory
- ruby example.rb
- irb ⇝ a playground/sandbox
  - exit to exit
  - "Interactive Ruby"

→ Interactive Ruby (IRB)
- sandbox (playground)
- exit or quit to leave irb
- known as REPL : Read Evaluate Print Loop
- control+c to force exit a program

2) Getting Started..

→ gets & gets.chomp.

gets ⇝ get String

⚠ default output includes \n
- to remove '\n', use .chomp

eg. name = gets.chomp

→ p method ⇝ gives more literal info.
- include " "
- include \n

3) Numbers & Booleans

→ Convert Numbers to String.
- .to_i ; .to_s ; to_f

→ Float Methods
- p 10.9.to_i ⇝ 10 (decimals are chopped off)
- p 10.9.floor ⇝ 10
- p 10.1.ceil ⇝ 11
- .abs
- .round ✓
- .round(2) ⇝ 10.10 (2 decimals)

→ .upto & .downto Methods

```
5.downto(1) {|i| puts ~~Countdown~~ i}
```

↳ 5
4
3
2
1 (inclusive).

```
1.upto(5) {|i| puts i}
```

1
2
3
4
5

→ .step method

```
1.step(10, 2) {|i| puts i}
```

↳ 1
3
5
7
9 ✓

```
0.step(10, 2) {|p| puts p}
```

0
2
4
6
8
10

4) Strings

→ Multiline Strings ← any char

```
words = <<MLS
```

~~~~ → Tab, enter, will be preserved.

MLS

p words ✓

→ Escape Characters

\" \' \n \t

↳ shown in p

→ Extract multiple characters from String

```
story = "abcdefg"
```

story[1,4] → bcde = story(1,4) ← extract 4 char
↑ from index 1

story[1..4] → bcde

↓ from index 1 → to index 4 (inclusive)
~~~~

→ .reverse , .include? , .empty? , .nil?

5) Methods & Conditionals
→ .respond_to? Method
→ case , !, unless, while, until

6) Range
→ .size
→ check if value exists in a range with .include?() or ===
"a"..."e".include?("b") → true
"a"..."e" === "b" → true
→ Generate Random Number with rand method
rand ~~.round(2)~~ →
↳ any float between 0 & 1.

+.round()
+(integer) → random number (integer) between 0 to given
* 100 → 0-100
(50..60) → random number between 50 & 60
(1..100) → random no. between 1 to 100.

8) Array
→ %w[] shorthand , space as delimiter
→ = Array.new(10, "a"..)
→ .fetch(index) to access ≈ ar[]
→ .values_at() Method
eg. [a1,2,3,4,5].values_at(0,2,4) → [1,3,5]
→ .slice() method
eg. [1,2,3,4,5].slice(2,3) → [3,4,5] ✓
≈ [] syntax.

→ .length , .size , .count , .first , .last

→ .push , << , .insert , .unshift

- ~~[1,2,3,4,5]~~.insert(index , value)
- .unshift() ⇒ add to start

→ .pop , .shift

→ <=> operator ⇒ -1 , 0 , 1 , nil

→ Convert Ranges to Arrays with the .to_a method

"A".."T".to_a

→ is_a? to check whether an object is made from a specific class

→ ri Program

- ri ——→ ruby class eg. String . Array
  - instance method eg. String.upcase , Array.shift
  - method eg. ri length

→ .each , .each_with_index , .map , .collect , .reverse , .sort ,
.break , .next , .concat , .max , .min , .include? , .index , .find_index
.select , .reject , .partition

→ a,b,c = [1,2,3] *

→ object pointers & Object Copies

- .object_id ⇒ object's location in memory
- .dup

→ Splat Arguments ⇒ allows any number of arguments.

- def sum(*numbers)

```
  sum = 0
  numbers.each{|num| sum += num}
  sum
end
sum(1,2,5,6,9,...) ✓
```

→ .any? , .all? method.

↳ [1,3,5,7,9].any? do |num|
  num.even?
end ⇝ false.

→ .find, .detect ⇝ ≈ select but only return the first result.

→ .uniq ,

→ .inject ≈ .reduce 0 if +, 1 if *

Array.inject(initial_value) do |stored_value, current_value|
  stored_value + current
end *

eg. [1,2,3,4,5].inject(0) do |s, c|
  puts "stored-value : #s"
  puts "current-value : #c"
  s+c
end ⇝ 15

↳ use it when you need a memory to store for each iteration

↳ method, argument, block provide.

[1,2,3,4,5].inject(1) do |s,c|
  puts "stored-value : #s"
  puts "current-value = #c"
  s*c
end
↳ 120

→ .flatten to remove internal arrays / unpack multidimensional array.

↳ Recursive; doesn't matter how many level of array.

→ .zip : combined array ~~into flash~~ based on same index positions.

↳ ["a",5,"c"].zip([1,2,3])

↳ [1,2,3].zip([4,5,6], ["A","B","C"]) ⇒ [[1,4,"A"],[...],[...]]

↳ def custom_zip(arr1, arr2)
  final = []
  arr1.each_with_index do |v,i|
    final << [value, arr2[i]]
  end
end final

→ .sample → return a random item from an array.
↳ ~~with~~ .sample(num) → return num items
↳ .sample(1) → always return in array form.
↳ Random order.

→ [1,2,3] * 2 → [1,2,3,1,2,3]
↳ def custom_multiply(array, number)
result = []
number.times { array.each { |element| result << element } }
result
end

→ [1,2,3] | [3,4,5] → [1,2,3,4,5]

↳ def custom_union(arr1, arr2)
arr1.dup.concat(arr2).uniq
end
↳ modify permanently
↳ X permanently permanately mutate.

→ [1,2,3,4] - [2,4,6,8] → [1,3]

↳ def custom_subtraction(arr1, arr2)
final = []
arr1.each { |value| final << value unless arr2.include?(value) }
↳ ! if
final.uniq
end
↳ ~~new = arr1.dup. each { |i|~~
new = arr1.dup
new.each { |i| new.delete(i) if arr2.include?(i) }
new.uniq

→ [1,2,3,4] & [2,4,6,8] ⟹ [2,4]

↳ def custom_intersection(arr1, arr2)
final = []
arr1.uniq.each { |e| final << e if arr2.include?(e) }
final
end

- Hash ⇄ Array
  - p hash.to_a → [[:k1, "v1"], [:k2, "v2"], ...]
  - p [[:k1, "v1"], [:k2, "v2"], ...].to_h → {:k1 => "v1", ...}
- .sort, .sort_by → sort by key/value.
  - ↳ return an array sorted by keys
    multi-D
- .key?, .value?,
- Hashes as method argument.
  - ↳ Solve wrong order issue.
  - ↳
```
def method(hash)
  hash[:k1] + hash[:k2] * hash[:k1]
end
```
- .delete(:key) → delete k-v pair.
- .select & .reject on Hash ≈ .each
  - ↳ .select {|k|, |v| ... }
  - ↳ criteria ∝ k/v
- .merge
  - ↳ hash1.merge(hash2) ← where duplicated ~~key~~ k-v preserved.
  - ↳
```
def custom_merge(hash1, hash2)
  new_hash = hash1.dup
  hash2.each {|k|, |v| new_hash[k] = v }
  new_hash
```
- def word_count(string)

```
  words = string.split(" ")
  count = Hash.new(0)
  words.each { |word| ~~word~~ count[word] += 1 }
  count
end
```

14) Blocks, Procs, & Lambdas

* value returned by 'puts' statement
  ↳ nil.

→ 'yield' keyword
  ↳ yield, a = yield, yield yield
    ↳ def who_am_i
        adjective = yield
        puts "I am very #{adjective}"
      end
      who_am_i { "handsome" } → I am very handsome.
      (implicit return & stored in 'adjective')
    ↳ don't include 'return' on yield block → else ERROR.
  ↳ yield if block_given?
  ↳ yield "argument"
    ↳ method { |arg| puts "yield with #{arg}" }
      ↳ yield with argument.

→ def method (arg)
    yield (arg)
  end
  method ("Kyle") { |n| puts "#{n} is handsome" }
    ↳ Kyle is handsome.

↳ def method (a1, a2, a3)
    yield (a1, a2, a3)
  end
  method (1, 2, 3) { |a, b, c| a+b+c } → 6

→ Enumerable.

↳ ['a','b','c'].all? { |word| word.length < 2 } → true.

```
def all?
  self.each do |i|                > arr = []
    condition = yield(i)
    condition ? arr << true : arr << false
  end
  puts arr.include?(false) ? false : true
end
```

↳ ['a','b','c'].any? ⇝ ↗

≈ none?

↳ [1,2,4,2].count { |x| x%2==0 } ⇒ 3

```
def count
  self.each do |i|                > counter = 0
    condition = yield(i)
    ~~condition ? arr << true : arr << false~~
    ~~counter+=1~~                ↳ counter += 1 if condition
  end
  puts counter
end
```

↳ [1,2,3,4,5].map { |i| i**2 } ⇝ [1,4,9,16,25]

```
def map
  arr = []                        > arr = []
  self.each do |i|
    condition = yield(i)
    ~~arr~~ arr << condition
  end
  arr
end
```

↳ [1,2,3,4,5].inject(0) { |p,c| p+c } ⇝ 15

```
def ~~inject(param)~~
  result = ~~param~~              ↳ method(param)
  ~~yield(a,b)~~
```

~~def~~ for i in 0..self.length-1
~~for self.each~~
yield(~~a,b~~) i, i+1)

⋮

→ Procs

↳ container for blocks.

↳ ∴ blocks is not an object, you can't call methods on it, or saved them into variables. → Procs / Lambda.

↳ cubes = Proc.new { |n| n ** 3 }

↳ when calling, prefix with & . eg. ~~acubes{~~ a.map(&cubes)

↳ * Array unpark:

~~a,b,c = f~~

a_cubes, b_c, c_c = [a,b,c].map { |arr| arr.map(&cubes) }

↳ Proc can be called directly with .call method.

↳ hi = Proc.new {puts "Hi there"}

hi.call → Hi there

5.times(&hi)

~~↳ to_proc~~

↳ Pass a Ruby Method as Proc

↳ * Helpful with iteration function like .map, .select, .reject

↳ p ["1", "2", "3"].map(&:to_i)

p [10, 14, 25].map(&:to_s)

p [1,2,3,4,5].select(&:even?)

p [1,2,3,4,5].reject(&:odd?)

↳ Methods with Proc Parameters. ★★

↳ def talk_about(name, &myproc)

puts "Let me tell you about #{name}."

myproc.call(name)

end

good = Proc.new {|name| puts "#{name} is a genius"}

bad = Proc.new {|name| puts "#{name} is a dolt"}

talk_about("Kyle", &good)

talk_about("Kyle", &bad)

→ Lambda
↳ strict regarding the arguments it gets & it needs.
↳ Return within lambda would exit lambda & the method continues.
↳ a = lambda { |n| n ** 2 }

→ Time Object (check previous note)

→ File Input & Output (use terminal)
↳ Read:
↳ File.open("file.txt").each do |line|
puts line
end
↳ ruby read_file.rb ✓

↳ Write to a Text file with Ruby
↳ File.open("file.txt", "w") do |file|
file.puts "a"
file.write "b"
file.puts "c"
end
↳ overwrite/create a new one : "w"
append to a file : "a"

→ At file.txt.
a
bc

↳ .rename ; .delete if File.exist('w')
↳ Command Line Arguments
↳ ARGV.each do |arg|
num = arg.to_i
puts "The square of #{num} is #{num ** 2}"
end
↳ ruby file.rb 3 5 7 9 10 (at CLI)
↳ The square of 3 is 9
⋮
↳ Load method
↳ load "file.rb" ↳ load "file.rb" if "a" > "b"
↳ load "./file.rb"

↳ require & require_relative methods.

↳ require "./file.rb" / require "./file"

↳ require_relative "end.rb" / require_relative "end"

↳ default (same directory)

require_relative "a/b/file.rb" → a → b → file.rb .

→ RegExp

↳ .scan ; .start_with? ; end_with? , .include? ; =~ ; /[]/ ;

.sub , .gsub ; ^ ; \

↳ ~~Rubular.com~~ rubular.com

→ [1-5].inject(0) {|p,c| p+c} → → 15

```
def inject(param)
  ~~for~~
  ~~self.each do |param|~~
  self.each do |i|
    ans = yield(param, i)
    param = ans
  end
  ans
end
```