

Learn to Code with Ruby 2.0

208) 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
```

```
5.times { |i| puts i } → 0  
                                1  
                                2
```

```
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 methods def, 'self' refer to the object.

```

class Bicycle

@@maker = "Bike Tech"

@@count = 0

→ or Bicycle.description

def self.description

"renenene pupu"

end

end

puts Bicycle.description → renenene 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  $\xleftarrow{\text{inherit}}$  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 .  
( e.g. 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
```

"yo, 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  $\rightsquigarrow$  "Yo, I'm Bob & I'm 48 year old"

p dan.introduce  $\rightsquigarrow$  "Yo, I'm Daniel & I'm 36 year old".

### 21.3) Check Inheritance Hierarchy

[ p Manager.ancestors  $\rightsquigarrow$  [Manager, Employee, Object, ...]

p Manager.superclass  $\rightsquigarrow$  Employee

puts Manager < Employee  $\rightsquigarrow$  true.]

### 21.4) .is-a? .instance-of? Predicate Methods.

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

[ puts bob.is-a?(Manager)  $\rightsquigarrow$  true  $\rightsquigarrow$  true if it's ancestors.

puts dan.is-a?(Employee)  $\rightsquigarrow$  true.

puts dan.is-a?(Worker)  $\rightsquigarrow$  true.

\* .instance-of?  $\rightsquigarrow$  true if its 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.

```
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
```

methods  
↳ to build on top of superclass, rather than rebuilding them, to adhere to DRY.

## 21.8) The 'super' Keyword #.

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

```
class Car
 def drive
 "Roon! Roon!"
 end
end
```

case ① with()

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

end

p ft.drive → Roon! Roon! Beep! Beep!

```
class Firetruck < Car
 def drive(speed) case ②
 super() + "Beep! Beep! I'm driving at #{}speed} miles per hr."
 end
end
```

ft = Firetruck.new  
p ft.drive(45)



## 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.

Γ class Player

def play-game

rand(1..100) > 50? "Winner!" : "Loser!"

end

end

bob = Player.new

boris = Player.new

def boris.play-game  
"Winner!"

→ Singleton Method.

→ 100% winner

end

p bob.play-game → "Winner!" / "Loser!"

p boris.play-game → "Winner!"

→ singleton-methods

p boris.singleton-methods → [:play-game]

p bob & " → []

p boris.singleton-class → #<Class: ... >

- ↪ 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

Γ class Candidate

attr\_accessor :name, :age, :occupation, :hobby

def initialize(name, details)

@name = name

@age = details[:age]

@occupation = details[:occupation]

end  
@hobby = details[:hobby]

~~#~~ 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 II.

\* def initialize(name, details = {}) <sup>default</sup>  
~~default = {age: 35, hobby: "chess", occupation: "Candidate", hobby: "Running"}~~

\* senator = Candidate.new("Kyle", {<sup>optional</sup>hobby: "chess", <sup>optional</sup>age: "25"})

\* defaults = {age: 35, occupation: "Candidate", hobby: "Running"}

defaults merge!(details)

↳ To return valid stuffs rather than

@age = defaults[:age]

nil if key is not given

@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  
without arguments method arguments to it

super with()  
without arguments - Call the same method in the superclass & pass no  
arguments to it.

\* ~~#~~ Subclass < Superclass

## 22) Reading & Writing Documentation

22.1) [www.rubydocs.info/stdlib/core](http://www.rubydocs.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 --

# inn

attr --

# inn

def --

→ rdoc album.rb

↳ 4  
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 where 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 e.g. 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 > atom-runner → setting to look for its spec
    - Install (Packaged)
    - `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 (> decimals)

→ .upto & .downTo Methods

5. downTo (1) { |i| puts ~~last~~ 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

words = <<MLS  
any char

~~~~~

→ Tab, enter, will be preserved.

MLS

p words ✓

→ Escape Characters

\ " \ ' \n \t

↳ shown in p

→ Extract multiple characters from String

story = " abcde fg " from index 1

story[1, 4] → b c d e = story(1, 4) extract 4 char

story[1..4] → b c d e

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 [1,2,3,4,5].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? → check whether an object is made from a specific class
- ri Program
  - ↳ ri → ruby class e.g. String, Array
  - ↳ instance method e.g. String.upcase, Array.shift
  - ↳ method e.g. 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      ↳ if +  
                                ↓ f\*

Array.inject(initial-value) do |stored-value, current-value|  
    stored-value + current  
end                              ↓ \*

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

    puts "stored-value: #s"  
    puts "current-value: #c"  
    end  
                                    ↓  
                                    ↓ 120

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

↳ method , argument , block  
provide.

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

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

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

↳ ["a","b","c"].zip([1,2,3])

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

↳ def custom\_zip(arr1, arr2)  
    final = []  
    arr1.each\_with\_index do |v,i|  
        final <- [v, arr2[i]]  
    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

↳ ~~modifies~~  
permanently

↳ X ~~permanently~~  
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) }

  final.uniq

↳ ! if

end

↳ ~~arr1~~  
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  $\leftrightarrow$  Array  
 $p \text{ hash.to\_a} \rightarrow [[\text{:k1, "v1"}], [\text{:k2, "v2"}], \dots]$   
 $P [[\text{:k1, "v1"}], [\text{:k2, "v2"}], \dots], \text{to\_h} \rightarrow \{\text{:k1} \rightarrow \text{"v1"}, \dots\}$
- .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)  
 $\quad \text{hash}[\text{:k1}] + \text{hash}[\text{:k2}] * \text{hash}[\text{:k1}]$   
 end
- .delete(:key) → delete k-v pair.
- .select & .reject on Hash ≈ .each  
 ↳ .select { |k1, v1| ... }  
 ↳ criteria on k/v
- .merge  
 ↳ hash1.merge(hash2)  $\leftarrow$  where duplicated key k-v preserved.  
 ↳ def custom\_merge(hash1, hash2)  
 $\quad \text{new\_hash} = \text{hash1.dup}$   
 $\quad \text{hash2.each} \{ |\text{k1}, \text{v1}| \text{new\_hash}[\text{k1}] = \text{v1} \}$   
 $\quad \text{new\_hash}$
- def word\_count(string)  
 $\quad \text{words} = \text{string.split(" ")}$   
 $\quad \text{count} = \text{Hash.new(0)}$   
 $\quad \text{words.each} \{ |\text{word}| \text{count}[\text{word}] += 1 \}$   
 $\quad \text{count}$   
 end

## 14) Blocks, Proc, & Lambdas

value returned by "puts" statement  
→ nil.

→ "yield" keyword

→ yield, a=yield, yield

↳ def who\_am\_i:

adjective = yield

puts "I am very #{{adjective}}"

end

↳ who\_am\_i["handsome"] → I am very handsome.

→ 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") { |a| puts "#{{a}} 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|  $\rightarrow \text{arr} = []$

condition = yield(i)

condition ? arr << true : arr << false

end

puts arr.include?(false) ? false : true

end

↳ `[ 'a', 'b', 'c' ].any? == true`

? none?

↳ `[ 1, 2, 4, 2 ].count { |x| x % 2 == 0 }`  ⇒ 3

def count

self.each do |i|  $\rightarrow \text{counter} = 0$

condition = yield(i)

condition ? arr << true : arr << false

end

~~counter += 1~~  $\rightarrow \text{counter} + = 1$  if condition

puts counter

end

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

def map

$\rightarrow \text{arr} = []$

self.each do |i|

condition = yield(i)

~~arr~~  $\rightarrow \text{arr} << \text{condition}$

end

arr

end

~~for i in self.length - 1~~  
~~self.each~~  
 $\rightarrow \text{yield}(a, b) \rightarrow \text{arr}[i, i+1]$

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

def inject(param)

~~result = param~~

~~yield(a)~~

method(param)

## → Proc

- ↳ container for blocks.
- ↳ ∵ blocks is not an object, you can't call methods on it, or save them into variables. → Proc / Lambda.
- ↳ cubes = Proc.new { |n| n\*\*3}
- ↳ when calling, prefix with & . e.g. ~~a.cubes~~ a.map(&cubes)
- ↳ \* Array unpack:  
~~a,b,c = f~~  
a\_cubes, b\_cubes, c\_cubes = [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| }

## ~~Proc~~

- ↳ Pass a Ruby Method as Proc
- ↳ \* Helpful with iteration function like .map, .select, .reject
  - ↳ p [1, 2, 3].map { |i| i }
  - ↳ p [10, 14, 25].map { |s| s }
  - ↳ p [1, 2, 3, 4, 5].select { |e| e.even? }
  - ↳ p [1, 2, 3, 4, 5].reject { |o| o.odd? }
- ↳ Methods with Proc Parameters. ~~★~~
- ↳ def talk\_about(name, &my\_proc)  
    puts "Let me tell you about #{}{name}."  
    my\_proc.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"` → At file.txt.  
↳ `file.write "b"` ↗  
↳ `file.puts "c"` ↗  
↳ `end`

a  
bc

↳ overwrite/create a new one : "`w`"

append to a file : "`a`"

↳ `.rename ; .delete if File.exist?('name')`

↳ 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 CII)

↳ The square of 3 is 9

↳ Load method

↳ `load "file.rb"`

↳ `load "./file.rb"`

↳ `load "file.rb" if a>b"`

- ↳ 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" →  file.rb

## → Reg Exp

- ↳ .scan ; .start-with? ; end-with? ; .include? ; =~ ; /E/;
- ↳ .sub , .gsub ; ^ ; \
- ↳ ~~Rubyfor.com~~ rubular.com

→ [-] inject (o) { |p,c| p+c } → → 15

def inject (param)

~~self~~.each do |param|

self each do |i|

ans = yield (param, i)

param = ans

end

ans

end