# W3D1

## Ruby Review

---

## Lecture Norms

---

## Lecture Slides

- All of the lecture slides and code from the demo can be found in the `cohort-resources` repo on Github, under the branch 06-13-2022.

---

## Typical Lecture Format

\* Learning Objectives of the Day

\* Today's Material (2hr ~ 3hrs)

\* Kahoot! Quiz

---

## A Quick word about Questions

1. How relevant is it to everyone?

2. Will the answer help today?

3. How much space am I taking?

4. Am I just showing off?

---

## Participate!

\* Volunteer to answer questions posed by the lecturer

\* Don't be afraid of saying the wrong thing

---

## Let's get started

---

## Learning Objectives

- Explain the differences between blocks and procs

- Explain the differences between class and instance methods

- Write Ruby code using proper style and methodologies

---

## Agenda

\* Blocks/Procs (Slides + Demo)

\* Class/Instance Methods (Slides + Demo)

\* Kahoot! 1

\* Ruby Style (Slides)

\* Kahoot! 2

---

## Blocks & Procs

\* \*\*Blocks\*\* - \*Code snippets\* contained by `{ }` or `do..end` that are passed to methods.

\* \*\*Procs\*\* - \*Objects\* that wrap blocks, and allow us to call/invoke the block.

\* Defining a method parameter with an ampersand like `&prc` takes a \_block\_ passed to the method and converts it into a \_proc\_ so that it can be called within the method (`prc.call`)

\* A method can only be invoked with one block, therefore there can only be one parameter with an ampersand(`&`)

---

## Blocks & Procs Demo

---

## Instance vs Class methods

---

## Instance methods

\* Methods that are accessible to an \_instance\_ of a class - `self` is set to the instance in the context of the method

\* Used to perform operations/procedures relating to the instance and its properties (i.e. instance variables)

\* Defined using `def <method\_name>` within the class definition

---

## Class methods

\* Methods that are accessible to the \_class itself\_ - `self` is set to the class in the context of the method

\* Used to perform operations/procedures relating to the class as a whole and its properties (i.e. class variables)

\* Used to perform general operations/procedures that instances may need but that don't relate to their specific properties (i.e. instance variables)

\* \*\*Factory Methods\*\* are special class methods used to create a specific \_type\_ of instance of that class

\* Defined using `def self.<method\_name>` within the class definition.

---

## Instance/Class Methods Demo

---

---

## Ruby Code Style

---

`"Imagine you are writing an email. You are in front of the

computer. You are operating the computer, clicking a mouse and typing

on a keyboard, but the message will be sent to a human over the

internet. So you are working before the computer, but with a human

behind the computer. Most of the tasks we do are for humans."`

Yukihiro Matsumoto, Chief Designer of Ruby

---

### Guidelines

\* Style is essential - \*\*follow the style guide!\*\*

\* During interviews, poor code style is a dead giveaway that someone lacks experience.

\* Good ruby code should \*\*read like English\*\*.

---

### Rules

#### Snake, camel case, screaming snake in Ruby

```ruby

variable\_names = 🐍 "snake\_case"

ClassNames = 🐫 "CamelCase"

CONSTANTS = 😱 🐍"SCREAMING\_SNAKE\_AHHHH!"

```

---

#### `{}` > `do .. end` for one lines

```ruby

[1, 2, 3].each do |el|

# multiple

# lines

# ...

end

[1, 2, 3].each { |el| puts el } # single line!

```

---

#### single line `if` / `unless`

```ruby

if truthy\_thing?

# multiple

# lines

end

puts "truthy!" if truthy\_thing? # single line!

```

---

#### 80 character limit for lines

```ruby

rant = "You could write really, really, really long lines in Ruby but that is bad code style should be avoided. Do not have more than 80 characters on one line."

```

---

#### Interpolation > Concatenation

```ruby

name = "Markov"

puts "Why, hello there " + name + ", have a lovely day!"

puts "Why, hello there #{name}, have a lovely day!"

```

---

#### Use `!` for mutating methods, `?` for boolean methods

```ruby

"abc".upcase!

[1, 2, 3].reverse!

5.even?

"abc".is\_a?(String)

```

---

#### Prefer block parameters over `yield`

More explicit:

```ruby

def do\_the\_thing(x, &prc)

prc.call(x)

end

```

Less explicit:

```ruby

def do\_the\_thing(x)

yield(x)

end

```

---

### No Code Golf

```ruby

def digitize(num)

[].tap { |digs| (num, dig = num.divmod(10); digs.unshift(dig)) until num.zero? }

end

def ping\_pong\_filter(arr)

arr.tap { |a| a.reject!.with\_index { |el, i| i.even? }.reverse! until a.length == 1 }[0]

end

def increase\_responsiveness?(arr)

arr.each\_cons(3).any? { |a| a.all? { |n| n > 0 } || a.all? { |n| n < 0 } }

end

def neighbor\_sum(arr)

[0, arr, 0].flatten.tap { |a| return a.map.with\_index { |\_, i| a[i - 1] + a[i + 1] if a[i + 1] }[1..-2] }

end

```

---

#### Use proper spacing and indentation

```rb

def my\_function

system("clear")

puts "Let's do this thing!"

loop do

arr = []

until arr.length == 6

until letter

letter = gets.chomp

if not\_valid?(letter)

puts "Wrong."

letter = nil

end

arr << letter

end

end

puts "The result is: #{arr}"

end

```

Find the bug in this code

---

```rb

def my\_function

system("clear")

puts "Let's do this thing!"

loop do

arr = []

until arr.length == 6

until letter

letter = gets.chomp

if not\_valid?(letter)

puts "Wrong."

letter = nil

end

arr << letter

end

end

puts "The result is: #{arr}"

end

```

---

#### No one-letter variable names

Exhibit A:

```rb

def do\_something(f, s)

x = nil

until x == s

x = @y.z(f)

a(x) unless v?(x)

x = nil

end

puts "Congratulations!"

end

```

---

#### No one-letter variable names 2

Exhibit B:

```rb

def do\_something(prompt, answer)

input = nil

until input == answer

input = @current\_player.guess(prompt)

unless valid\_guess?(input)

alert\_invalid\_guess(input)

input = nil

end

end

puts "Congratulations!"

end

```

Which one would YOU rather your teammate wrote?

---

### Caveat

\* Exception to this is loops.

\* But even for loops, good to use more meaningful names.

\* Especially when using nested loops.

---

## Recap 1

\* Adhere to case conventions for `variable\_names`, `ClassNames`, and `CONSTANTS`

\* Use `do... end` for multi-line blocks and `{}` for single-line blocks.

\* If you can keep an `if` or `unless` statement to one line, do it!

\* Try to keep your lines shorter than 80 characters

---

## Recap 2

\* Interpolation > Concatenation

\* `!` for mutating methods, `?` for boolean methods

\* `prc.call` > `yield`

\* Use proper spacing and indentation!

\* Name your variables so your code reads like English!

Note: These rules will differ by language. Stick to the conventions for that language. Your coworkers will thank you for it.

---

## [Kahoot!](https://play.kahoot.it/v2/?quizId=25045d9c-c373-4b00-ab7c-023d6c3542fe)

---

Thank you and have fun today!