Programming for Everybody

8. Blocks, Procs and Lambdas

le wagon

Blocks recap

blocks are chunks of code between curly braces **{}** or between the keywords **do** and **end** that we can associate with method invocations

```
puts [1, 2, 3].map { |num| num ** 2 }
```

block

prints out [1, 4, 9]

Yield

we can code custom methods that accept “external blocks” by typing the **yield** keyword within our method

if we call that method followed by a block, whichever code is on that block will replace the yield keyword inside the method

```
def welcome_message
    print “Welcome!”
    yield
    puts “ Enjoy!”
end

welcome_message { print “ Today we’ll learn about procs.” }
```

(prints out “Welcome! Today we’ll learn about procs. Enjoy!”)

Procs & lambdas

if we want to be able to reuse a block to keep our code DRY, we need to create a **proc** or a **lambda**

procs and lambdas are no more than **blocks saved to a variable**

Procs

a **proc** is a block saved to a variable

it does not care for the number of arguments it gets

we can call it directly through the **.call** method or we can pass it into a method as an argument

when passed into a method, a proc does not give the control back to that method after returning

Proc syntax

1. Creating & calling a proc

```
today_lecture_proc = Proc.new do
    puts “Today we’ll learn about procs.”
end


today_lecture_proc.call
```

2. Passing a proc to a method

```
def welcome_message
    print “Welcome!”
    yield
end

welcome_message(&today_lecture_proc)
```

we pass the proc as an argument of the method like this:
with an & before its name

(prints out “Welcome! Today we’ll learn about procs.”)

Lambdas

a **lambda** is a block saved to a variable

it checks the number of arguments it gets

we can call it directly through the **.call** method or we can pass it into a method as an argument

when passed into a method, a lamba gives the control back to that method after returning

Lambda syntax

1. Creating & calling a lambda

```
today_lecture_lambda = lambda do
    puts “Today we’ll learn about lambdas.”
end


today_lecture_lambda.call
```

2. Passing a lambda to a method

```
def welcome_message
    print “Welcome!”
    yield
end

welcome_message(&today_lecture_lambda)
```

(prints out “Welcome! Today we’ll learn about lambdas.”)

Method names as procs

we can call a method by passing its name as a symbol (ex :to_i, :to_s, :capitalize, etc.) preceded by an **&** -> this ends up actually being a proc!

names = [“mariana”, “mark”, “peter”]

puts names.each { |name| name.capitalize! }

puts names.each(&:capitalize!)

note the colon for symbol and the & that transforms it into a proc

prints out Mariana Mark Peter

Thank you! :)

le wagon