codecademy

Blocks, Procs, and Lambdas

Ruby .call Method

In Ruby, a *proc* and a *lambda* can be called directly using the `.call` method.

```
proc_test = Proc.new { puts "I am the proc method!" }
lambda_test = lambda { puts "I am the lambda method!"}

proc_test.call # => I am the proc method!
lambda_test.call # => I am the lambda method!

#The following code would result in "I am the proc method!" and "I am the lambda method!" printed to the console respectively, once the proc, proc_test, and the lambda, lambda_test, are called.
```

Ruby lambda

In Ruby, a *lambda* is an object similar to a *proc*. Unlike a *proc*, a *lambda* requires a specific number of arguments passed to it, and it `return`s to its calling method rather than returning immediately.

```
def proc_demo_method
  proc_demo = Proc.new { return "Only I print!" }
  proc_demo.call
  "But what about me?" # Never reached
end

puts proc_demo_method
# Output
# Only I print!

# (Notice that the proc breaks out of the method when it returns the value.)

def lambda_demo_method
  lambda_demo = lambda { return "Will I print?" }
  lambda_demo.call
  "Sorry - it's me that's printed."
end

puts lambda_demo_method
# Output
# Sorry - it's me that's printed.

# (Notice that the lambda returns back to the method in order to complete it.)
```

Ruby .collect Method

In Ruby, the `.collect` array method takes a block and applies the expression in the block to every element of an array.

```
first_arr = [3, 4, 5]
second_arr = first_arr.collect { |num| num * 5 }

print second_arr #Output => [15, 20, 25]

# In this example, the .collect method is used to multiply each number within first_arr by 5. The outcome is then saved inside of the second_arr variable and printed to the console. The original first_arr is left unchanged.
```

Ruby yield Keyword

In Ruby, the `yield` keyword is used to transfer control from a method to a block and then back to the method once executed.

```
def yield_test
  puts "I'm inside the method."
  yield
  puts "I'm also inside the method."
end

yield_test { puts ">>> I'm butting into the m
ethod!" }
#Output
# I'm inside the method.
# >>> I'm butting into the method.
# I'm also inside the method.
```

Ruby proc

In Ruby, a *proc* is an instance of the Proc class and is similar to a block. As opposed to a block, a *proc* is a Ruby object which can be stored in a variable and therefore reused many times throughout a program.

```
square = Proc.new { |x| x ** 2 }
# A proc is defined by calling Proc.new follo
wed by a block.

[2, 4, 6].collect!(&square)
# When passing a proc to a method, an & is us
ed to convert the proc into a block.

puts [2, 4, 6].collect!(&square)
# => [4, 16, 36]
```