Programming for Everybody

8. Procs and Lambdas

le wagon

Blocks Recap

Blocks (a.k.a anonymous/nameless functions) are block of codes executed inside a method (like `each`, `sort`, etc.)

They are similar to methods, but they don't need a name or to be assigned, because they are already inside another function

```ruby
names = ["gabriele", "mariana", "shannon"]

names.each do |name|
  reversed_name = name.reverse
  puts reversed_name.upcase
end
```

← Nameless block of code

Yield

What if you want to write a custom method that accepts a block, like it happens with `each` and the other standard functions?

Use the **`yield`** keyword!

When your method will be called with a block, that block will replace your `yield` keyword inside the method

```ruby
def welcome_message
 puts “Welcome!”
 yield
end
```

```ruby
welcome_message do
 print “Today we’ll see procs”
 puts “ and lambdas!
end
```

Everything inside this block replaces the yield!

Procs

Until now we've been writing different blocks of code inside other methods, but never stored them in variables

So, what if you want to store and reuse them?

Procs are made exactly for that!

```
def welcome_message
 puts "Welcome!"
 yield
end
```

```
today_lecture = Proc.new do
 print "Today we'll see procs"
 puts " and lambdas!
end

welcome_message(&today_lecture)
```

Pass the Proc like this (with an & before)!

Procs with Symbols

In Ruby, a *method <u>name</u> can be called and passed with a symbol*, like `:to_i`, `:to_s`, `:capitalize`, etc.

And what if we want to pass this method as a proc?

We can easily do it by converting the symbol to a proc!

```ruby
names = [“gabriele”, “mariana”, “shannon”]
```

~~`names.map! { |name| name.capitalize }`~~

```ruby
names.map!(&:capitalize)
```


Note the colon, we are using a symbol here!

Lambdas

With the exception of a bit of syntax and a few behaviours,
lambdas are identical to procs

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

```
today_lecture_proc = Proc.new  do
 print “Today we’ll see procs”
 puts “ and lambdas!
end

today_lecture_lambda = lambda  do
 print “Today we’ll see procs”
 puts “ and lambdas!
end

welcome_message(&today_lecture_lambda)
```

Let’s see some more live examples!

Thank you! :)

le wagon