Methods, Blocks , Procs and Lambdas

León Jaramillo

softserve

Methods, Blocks , Procs and Lambdas

- Ruby supports structured programming, allowing us using code constructions such as **methods**.
- Methods (and their cousins, functions) are probably the most common way of code reuse.
- Although in the language documentation the terms *function* and *method* are often used interchangeably, one can say that in this language *functions* "don't exist", while the term *method* is used in the broadest scope.
- Despite the above affirmation and although Ruby is an OOP language, it has some constructs that support **Functional Programming** (FP, what is it?):
  - Methods
  - Blocks
  - Procs
  - Lambdas

Methods

- Defining a method

```
def method_name [([param_name [= default_val]]...)]
   expressions...
   [return expression]
end
```

- Some examples

```
def method_with_no_params
   expr..
end

def method_with_params (var1, var2)
   expr..
end

def method_with_def_vals (var1 = value1, var2 = value2)
   expr..
end
```

- Calling a method

```
just_the_method_name

method_with_params 2, 3

method_with_params(2,3)

Method(name="Ruby")
```

softserve

Blocks

- A block consists of one or more statements (of code).
- Normally, a block is named and enclosed in braces (`{}`).
- You would invoke a block using the `yield` keywork from a method of the same name.

```
block_name {
   one statement
   other statement
   ...
}
```

- So, this is how you can typically use the former block:

```
def block_name
   statements
   yield
   statements
end
```

- There are some special blocks, that can execute at the beginning and at the end of the execution of a script:

```
BEGIN {
   statements…
}
END {
   statements…
}
```

Procs

- A Proc encapsulates a Block in a variable.
- You can call it and do with it anything you would do with a typical variable.
- You can create a proc in different ways:

```
proc1 = Proc.new {|x| x**2 }
proc2 = proc {|x| x**2 }
```

- Then, you can call them in different ways too:

```
proc1.call(arg1, arg2)
proc2.call([arg1, arg2])
```

- Comparing them with lambdas, procs are more flexible with their parameters, and their return statements behave differently.

Lambdas

- A Lambda is a special type of Proc. And unlike procs, lambdas are present in my other programming languages.

- You can call then and do with them anything you would do with a typical variable, too.

- You can create a lambda in different ways:

```
l = lambda {|x, y| "x=#{x}, y=#{y}" }
l = -> (x,y) {"x=#{x}, y=#{y}" }
```

- Then you can call it, similarly than with procs (but considering parameters' restrictions):

```
l.call(1, 2)
```

- You should consider the differences between procs and lambdas before using them.

Homework

1. Write a function that receives a integer number as a parameter and writes in the screen if its an even or an odd number.
2. Write a function that returns the factorial of a given number, using iteration.
3. Write a function that returns the factorial of a given number, using recursion.
4. Write a function that returns true if a given number is prime, and false otherwise.
5. Write a function that receives a string and an integer number, returning an array which contains the words of that string whose size is greater that the received number.

Useful Resources

- Methods: https://www.tutorialspoint.com/ruby/ruby_methods.htm
- Blocks: https://www.tutorialspoint.com/ruby/ruby_blocks.htm
- Procs: https://ruby-doc.org/core-2.6/Proc.html
- Lambdas: https://scoutapm.com/blog/how-to-use-lambdas-in-ruby
- Video-tutorial: https://www.youtube.com/watch?v=92yuNm6Ts0c&ab_channel=JesusCastello


COLOMBIA
DC
Thanks! Any question?
softserve