

PROGRAMAÇÃO FUNCIONAL

Rafael Vieira Coelho

(rafaelvc2@gmail.com)

Ruby

Aprenda a programar
na linguagem mais divertida



C Casa do
Código

LUCAS SOUZA

PROGRAMAÇÃO FUNCIONAL

- Trata-se de um paradigma de programação que trata a computação como a avaliação de funções matemáticas.
- Enfatiza o uso das funções que não alteram estado, ao contrário do que ocorre em programação imperativa.
- Este paradigma foi fundamentado em 1930 com o lambda calculus, um sistema formal para investigar a definição de funções, aplicação delas e recursão.

Vamos tomar como exemplo o caso onde desejamos atribuir um determinado valor a uma variável se uma condição for verdadeira e outro valor caso a condição seja falsa. Podemos criar esse código em Ruby da seguinte maneira:

```
1 valor = nil

2 numero = "dois"

3

4 if numero == "um" then valor = 1

5 elsif numero == "dois" then valor = 2

6 else valor = 3

7 end

8

9 p valor # => 2
```


USANDO PROGRAMAÇÃO FUNCIONAL

```
2 valor = if numero == "um" then 1  
  
3         elsif numero == "dois" then 2  
  
4         else 3  
  
5     end  
  
6  
  
7 p valor # => 2
```

EM RUBY, TUDO É AVALIADO COMO UMA EXPRESSÃO

Até mesmo quando utilizamos um `for`, o seu resultado é uma expressão. Se percorrermos um `Array` com três elementos usando um `for` e multiplicarmos cada item por 2, por exemplo, podemos atribuir seu resultado a uma variável:

```
1 numeros = [1, 2, 3, 4]
2 novos_numeros = for n in numeros
3     n * 2
4 end
5
6 p novos_numeros # => [1, 2, 3, 4]
```

POR QUE NÃO RETORNAR?

Todos os métodos Ruby retornam sempre o resultado da última expressão declarada, por esse motivo, você não precisa explicitamente adicionar o `return` no final de cada método:

```
1 def boas_vindas(nome)
2     "Bem vindo: #{nome}"
3 end
```

FUNÇÕES ALTA ORDEM

Funções ou métodos são high-order quando tem a capacidade de receber outras funções como argumentos ou retornar funções como resultado. Em Ruby isto é feito usando blocos, lambdas e procs.

Blocos, lambdas e procs são um dos aspectos mais poderoso da linguagem Ruby, e também um dos que causam mais confusões para serem entendidos, isso porque Ruby possui quatro maneiras de lidar com high-order functions.

FUNÇÕES ALTA ORDEM: BLOCOS

Este é o método mais comum trabalhar com funções high-order em Ruby. Os blocos são muito utilizados e comuns quando percorremos coleções:

```
1 numeros = [1, 2, 3, 4]
3 numeros.each { |numero| p numero }

4 # => 1

5 # => 2

6 # => 3

7 # => 4
```


FUNÇÕES ALTA ORDEM: BLOCOS

```
numeros = [1, 2, 3, 4]
```

```
numeros.each { |numero| p numero }
```

- Na chamada do método **each()** a partir da variável **numeros** passamos uma função ou bloco de código como parâmetro.
- Internamente, o **each()** itera o array e executa o bloco de código recebido passando pra ele o parâmetro **numero**.
- O bloco de código está imprimindo a variável **numero**.

FUNÇÕES ALTA ORDEM: BLOCOS

- **each:** usado quando não importa o índice.
- **each_with_index:** usado quando se quer ter acesso ao dado e ao índice.
- **each_index:** usado quando se interessa apenas pelos índices.
- **map:** quando se quer mapear um array em outro array.
- **select:** quando se quer um subconjunto da coleção.
- **inject:** útil para fazer cálculos nos valores ou obter um resultado único.

EXERCÍCIO

- Ache exemplos práticos das funções de alta ordem citadas previamente: `each`, `each_with_index`, `each_index`, `map`, `select` e `inject`.

<https://ruby-doc.org/core-1.9.3/Enumerable.html>

USANDO EACH

- Vamos criar um método que filtra os livros por uma determinada categoria, itera cada um destes livros e executa um bloco de código que será passado para este método.

Adicionando o método livros_por_categoria na classe Biblioteca:

```
1 class Biblioteca
2
3   attr_accessor :livros #get/set
4   #attr_reader :livros get
5   #attr_writer :livros set
6
7   def initialize
8     @livros = {}
9   end
10
11   def adiciona(livro, isbn)
12     @livros[isbn] = livro
13   end
14
15   def to_s
16     for isbn in @livros.keys
17       puts '='*40
18       puts "ISBN: #{isbn}"
19       puts @livros[isbn]
20       puts '='*40
21     end
22   end
23
24   def livros_por_categoria(categoria)
25     livros_categoria = []
26     @livros.each {|isbn, livro| livros_categoria << livro if livro.categoria == categoria }
27     livros_categoria
28   end
29 end
```

Testando o método criado:

```
1 require_relative 'livro'
2 require_relative 'biblioteca'
3
4 biblioteca = Biblioteca.new
5
6 liv1 = Livro.new("Mauricio Aniche", "123454", "Suspense")
7 liv2 = Livro.new "Tárcio Zemel", "452565", "Biografia"
8 liv3 = Livro.new("Rogério Amaral", "1111", "Suspense")
9 liv4 = Livro.new "Alberto de Souza", "1232", "Biografia"
10
11 biblioteca.adiciona(liv1, 247)
12 biblioteca.adiciona(liv2, 248)
13 biblioteca.adiciona(liv3, 249)
14 biblioteca.adiciona(liv4, 250)
15
16 #puts biblioteca
17
18 livros_suspense = biblioteca.livros_por_categoria("Suspense")
19
20 livros_suspense.each {|livro|
21   puts livro
22   puts '*'*30
23 }
```

```
Autor: Mauricio Aniche
Páginas: 123454
Categoria: Suspense
#<Livro:0x00007fbe9787d090>
*****
Autor: Rogério Amaral
Páginas: 1111
Categoria: Suspense
#<Livro:0x00007fbe9787ce88>
*****
```


USANDO INJECT

- Vamos criar uma classe chamada Relatório na qual criaremos métodos que usam o inject.

MAS ANTES, FAREMOS O RELATÓRIO COM O EACH

```
1  require_relative 'biblioteca'
2
3
4  class Relatorio
5
6      def initialize(biblioteca)
7          @biblioteca = biblioteca
8      end
9
10     def total()
11         soma = 0.0
12
13         @biblioteca.livros.values.each { |livro|
14             soma += livro.valor
15         }
16         soma
17     end
18
19 end
```

MAS ANTES, FAREMOS O RELATÓRIO COM O EACH

```
1 require_relative 'livro'
2 require_relative 'biblioteca'
3 require_relative 'relatorio'
4
5 biblioteca = Biblioteca.new
6
7 liv1 = Livro.new("Mauricio Aniche", "123454", "Suspense", 198.89)
8 liv2 = Livro.new "Tárcio Zemel", "452565", "Biografia", 53.21
9 liv3 = Livro.new("Rogério Amaral", "1111", "Suspense", 12.90)
10 liv4 = Livro.new "Alberto de Souza", "1232", "Biografia", 135.99
11
12 biblioteca.adiciona(liv1, 247)
13 biblioteca.adiciona(liv2, 248)
14 biblioteca.adiciona(liv3, 249)
15 biblioteca.adiciona(liv4, 250)
16
17 #puts biblioteca
18
19 relatorio = Relatorio.new(biblioteca)
20 puts 'TOTAL: ', relatorio.total
```

```
TOTAL:
400.99
[Finished in 0.1s]
```


USANDO INJECT

- Podemos melhorar o código usando o inject

- O **inject()** recebe como primeiro parâmetro um valor que será o acumulador (**soma**), inicializado em 0
- O segundo parâmetro recebemos cada livro do array de livros do objeto Biblioteca (**livro**)
- A cada iteração somamos o valor do objeto Livro a variável acumuladora **total** (retornado no final)

```
1  require_relative 'biblioteca'
2
3
4  class Relatorio
5
6      def initialize(biblioteca)
7          @biblioteca = biblioteca
8      end
9
10     def total()
11         @biblioteca.livros.values.inject(0) { |soma, livro|
12             soma += livro.valor
13         }
14     end
15
16 end
```

USANDO MAP

- A classe Relatorio precisa agora de um método que retorne o título de todos os livros que possuímos no objeto Biblioteca.

CRIAREMOS O OBTEM_TITULOS COM O EACH

```
1  require_relative 'biblioteca'
2
3
4  class Relatorio
5
6      def initialize(biblioteca)
7          @biblioteca = biblioteca
8      end
9
10     def total
11         @biblioteca.livros.values.inject(0) { |soma, livro|
12             soma += livro.valor
13         }
14     end
15
16     def obtem_titulos
17         titulos = []
18
19         @biblioteca.livros.values.each do |livro|
20             titulos << livro.titulo
21         end
22         titulos
23     end
24 end
25
```

CRIAREMOS O OBTEM_TITULOS COM O EACH

```
1  require_relative 'livro'
2  require_relative 'biblioteca'
3  require_relative 'relatorio'
4
5  biblioteca = Biblioteca.new
6
7  liv1 = Livro.new("Stephen King", "123454", "Suspense", 198.89, "A Torre Negra")
8  liv2 = Livro.new "André Agassi", "452565", "Biografia", 53.21, "Minha Vida"
9  liv3 = Livro.new("Rafael Coelho", "1111", "Suspense", 12.90, "Sra. Fantini")
10 liv4 = Livro.new "Stephen King", "1232", "Suspense", 135.99, "It"
11
12 biblioteca.adiciona(liv1, 247)
13 biblioteca.adiciona(liv2, 248)
14 biblioteca.adiciona(liv3, 249)
15 biblioteca.adiciona(liv4, 250)
16
17 #puts biblioteca
18
19 relatorio = Relatorio.new(biblioteca)
20 puts 'Títulos: \n', relatorio.obtem_titulos
21
```

```
Títulos:
A Torre Negra
Minha Vida
Sra. Fantini
It
[Finished in 0.1s]
```

USANDO MAP

- Podemos reduzir o código através do método `map` no método **`obtem_titulos()`**.

```
1  require_relative 'biblioteca'
2
3
4  class Relatorio
5
6      def initialize(biblioteca)
7          @biblioteca = biblioteca
8      end
9
10     def total
11         @biblioteca.livros.values.inject(0) { |soma, livro|
12             soma += livro.valor
13         }
14     end
15
16     def obtem_titulos
17         @biblioteca.livros.values.map { |livro| livro.titulo }
18     end
19 end
```


USANDO MAP

- Ou ainda mais reduzido. O character **&** invoca um método **to_proc()** no objeto, e passa este bloco para o método map.

```
1  require_relative 'biblioteca'
2
3
4  class Relatorio
5
6      def initialize(biblioteca)
7          @biblioteca = biblioteca
8      end
9
10     def total
11         @biblioteca.livros.values.inject(0) { |soma, livro|
12             soma += livro.valor
13         }
14     end
15
16     def obtem_titulos
17         @biblioteca.livros.values.map &:titulo
18     end
19 end
```

FUNÇÕES ALTA ORDEM: PROCS

- Podemos receber um bloco de código como argumento do tipo **Proc**.

```
1 def metodo_comum(parametro, metodo1, metodo2)
2   metodo1.call(parametro)
3   metodo2.call(parametro)
4 end
5
6 parametro = 10
7
8 metodo1 = Proc.new do |parametro|
9   puts parametro + 10
10 end
11
12 metodo2 = Proc.new do |parametro|
13   puts parametro - 10
14 end
15
16 metodo_comum(parametro, metodo1, metodo2)

```

```
20
0
[Finished in 0.2s]
```

O & no método1 ou metodo2 (proc) transforma ele em bloco para ser usado em um each, por exemplo.

FUNÇÕES ALTA ORDEM: LAMBDA

- O termo **lambda** é originário de um cálculo criado em 1930 usado para investigar as fundações da matemática.
- Facilita o cômputo de funções através de sua semântica simplificada.
- São também conhecidas como **funções anônimas**.

FUNÇÕES ALTA ORDEM: LAMBDA

```
1  #lambda faz a verificação dos parâmetros
2
3  #lambda quando tem return, termina a sua execução
4  # e não retorna nada
5  metodo = lambda do |parametro|
6      puts parametro + 10
7  end
8
9  #converte lambda em Proc
10 [1,2,3,4,5].each(&metodo)
11
12 #chamamento normal
13 metodo.call(10)
```

11

12

13

14

15

20

[Finished in 0.1s]