# **Collections**

#### Conteúdo

O conteúdo que será abordado na aula:

* Collection
* Manipulando Collections
  + Array
  + Hash
* Iterações

Nesta aula você aprenderá o que são collections e como manipulá-las.

#### O que são Collections

Na programação, **collection** representa um **conjunto de dados semelhantes** em uma única unidade.

**Ex**: Um amigo tem uma grande quantidade de livros sobre programação e a fim de guardá-los de forma organizada os colocou dentro de uma caixa com o rótulo “Livros de Programação”.  
Note que nossa caixa é uma representação dos livros com conteúdo de Programação. Uma collection é exatamente isso, um local onde concentramos uma quantidade de itens semelhantes.

Agora, sempre que meu amigo precisar rever um livro de programação, ele saberá onde encontrá-lo!

Dois tipos de collections bastante utilizados na linguagem Ruby são Array e Hash, os quais conhecemos na segunda aula deste curso.

#### Manipulando Collections

### Array

Existem várias maneiras de manipular arrays. Abaixo encontram-se algumas muito úteis para todo programador.

**Criando um Array**

1- Crie um array vazio.

|  |  |
| --- | --- |
| 1  2 | irb  estados = [] |

Collections podem ter zero ou mais elementos.

**Adicionando itens**

1- Insira um novo item ao array **estados**.

|  |  |
| --- | --- |
| 1 | estados.push('Espírito Santo') |

O push sempre irá adicionar itens de forma sequencial.

2- Também é possível inserir vários elementos de uma só vez.

|  |  |
| --- | --- |
| 1 | estados.push('Minas Gerais', 'Rio de Janeiro', 'São Paulo') |

3- Veja o array **estados** com a instrução

|  |  |
| --- | --- |
| 1 | puts estados |

4- Para manter nossa coleção organizada em ordem alfabética ao inserir os itens ‘Acre’ e ‘Amapá’, devemos especificar que eles ocuparão as primeiras posições do array. Para isso contamos com o **insert**.

|  |  |
| --- | --- |
| 1 | estados.insert(0, 'Acre', 'Amapá') |

Primeiro é passado o valor do índice onde a instrução será aplicada  acompanhado por um ou mais itens a serem adicionados.

5- Exiba o array **estados**

|  |  |
| --- | --- |
| 1 | puts estados |

Os elementos Acre e Amapá tornaram-se os primeiros do array.

**Acessando elementos**

Como já vimos na segunda aula, o item de um array pode ser acessado pelo valor de seu index.

1- Recupere o segundo elemento do array **estados**

|  |  |
| --- | --- |
| 1 | estados[1] |

Saiba que o primeiro elemento não inicia no índice 1, mas sim no 0.

2- Você também pode acessar índices através de intervalos

|  |  |
| --- | --- |
| 1 | estados[2..5] |

Retorna os itens dos índices 2, 3, 4 e 5

Utilizando números negativos conseguimos recuperar elementos a partir do ultimo item do array, de forma regressiva. O número -1 representa o ultimo elemento.

3- Adquira o penultimo elemento de **estados**

|  |  |
| --- | --- |
| 1 | estados[-2] |

4- Também funciona com intervalos

|  |  |
| --- | --- |
| 1 | estados[-3..-1] |

5- Uma forma muito intuitiva e natural de recuperar o primeiro item é usar **first**

|  |  |
| --- | --- |
| 1 | estados.first |

6- Seguindo a mesma ideia, use **last**para o último

|  |  |
| --- | --- |
| 1 | estados.last |

**Obtendo informações**

1- Para saber a quantidade de itens em um Array você pode utilizar qualquer uma destas duas instruções

|  |  |
| --- | --- |
| 1  2 | estados.count  estados.length |

2- Descubra se o array está vazio

|  |  |
| --- | --- |
| 1 | estados.empty? |

O resultado será verdadeiro ou falso

3- Verifique se um item específico está presente

|  |  |
| --- | --- |
| 1 | estados.include?('São Paulo') |

Igual ao empty, também resulta um valor verdadeiro ou falso

**Excluindo elementos**

1- Remova um item através de seu índice

|  |  |
| --- | --- |
| 1 | estados.delete\_at(2) |

2- Exclua o ultimo item do array **estados**

|  |  |
| --- | --- |
| 1 | estados.pop |

3- Para excluir o primeiro item faça

|  |  |
| --- | --- |
| 1 | estados.shift |

### Hash

A seguir veja exemplos importantíssimos sobre manipulação de Hashes

**Novo Hash**

1- Crie um hash vazio

|  |  |
| --- | --- |
| 1 | capitais = Hash.new |

2- Um Hash também pode ser iniciado com vários pares de chave-valor

|  |  |
| --- | --- |
| 1 | capitais = { acre: 'Rio Branco', sao\_paulo: 'São Paulo'} |

**3-** A chave de um Hash pode ser qualquer tipo de dado

|  |  |
| --- | --- |
| 1 | hash = {1 => 'Chave do tipo inteiro', true => 'Chave do tipo booleano, [1,2,3] => 'Chave do tipo array'} |

**Adicionando itens**

1- Adicione um novo item ao hash **estados**

|  |  |
| --- | --- |
| 1 | capitais[:minas\_gerais] = "Belo Horizonte" |

2- Acesse a capital que acabamos de inserir utilizando sua chave

|  |  |
| --- | --- |
| 1 | capitais[:minas\_gerais] |

De forma sucinta, a chave é o index de nossos itens

3- Para retornar todas as chaves de um hash

|  |  |
| --- | --- |
| 1 | capitais.keys |

4- Agora, todos os valores de um hash

|  |  |
| --- | --- |
| 1 | capitais.values |

**Exclusão**

1- Remova um elemento chave-valor

|  |  |
| --- | --- |
| 1 | capitais.delete(:acre) |

**Obtendo informações**

1- Descubra o tamanho do hash

|  |  |
| --- | --- |
| 1 | capitais.size |

2- Verifique se o Hash está vazio

|  |  |
| --- | --- |
| 1 | capitais.empty? |

### Iterações

Agora, você será apresentado a três novas estruturas de repetição utilizadas para trabalhar com **collections**

### Each

Percorre uma coleção de forma parecida ao**for,**porém, não sobrescrevendo o valor de variáveis fora da estrutura de repetição.

### Array

1- Adicione o seguinte código a um programa chamado **each\_array.rb**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | names = ['Joãozinho', 'Manoel', 'Juca']    name = 'Leonardo Scorza'    names.each do |name|   puts name  end    puts name |

Ao executar o programa perceba que não foi alterado o valor da  variável name, definida antes da estrutura de repetição.

### Hash

1- Crie um arquivo chamado  **each\_hash.rb**com o seguinte código

|  |  |
| --- | --- |
| 1  2  3  4  5 | aulas = {'Aula 1 ' => 'liberada', 'Aula 2 ' => 'liberada', 'Aula 3 ' => 'liberada', 'Aula 4 ' => 'liberada', 'Aula 5 ' => 'em breve'}    aulas.each do |key, value|   puts "#{key} #{value}"  end |

Em cada vez que a estrutura percorre o hash, o elemento atual é representado por key e value.

**Map**

Cria um array baseando-se em valores de outro array existente.

1- Crie um arquivo chamado **map.rb**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | array = [1, 2, 3, 4]    # \n é uma quebra de linha  puts "\n Executando .map multiplicando cada item por 2"  # .map não altera o conteúdo do array original  new\_array = array.map do |a|               a \* 2             end    puts "\n Array Original"  puts " #{array}"    puts "\n Novo Array"  puts " #{new\_array}"    puts "\n Executando .map! multiplicando cada item por 2"  # .map! força que o conteúdo do array original seja alterado  array.map! do |a|   a \* 2  end    puts "\n Array Original"  puts " #{array}"  puts '' |

Como vimos neste exemplo, podemos forçar que o array original seja alterado utilizando map!

**Select**

Realiza uma seleção de elementos presentes em uma collection através de uma condição pré definida. Traz como resultado somente os valores que passam nesta condição.

**Array**

1- Crie um arquivo chamado **select\_array.rb**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | array = [1, 2, 3, 4, 5, 6]    selection = array.select do |a|               a >= 4             end    puts selection |

A condição para que um item do array seja selecionado é que seu valor seja maior ou igual a 4.

**Hash**

1- Crie um arquivo chamado **select\_hash.rb**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | hash = {0 => 'zero', 1 => 'um', 2 => 'dois', 3 => 'tres'}    puts 'Selecionando keys com valor maior que 0'  selection\_key = hash.select do |key, value|                   key > 0                 end    puts selection\_key |

Veja que dentro de um Hash podemos fazer uma seleção por chave ou valor.

### Missões especiais

**Missão 1**

Utilizando uma collection do tipo Array, escreva um programa que receba 3 números e no final exiba o resultado de cada um deles elevado a segunda potência.

**Missão 2**

Crie uma collection do tipo Hash e permita que o usuário crie três elementos informando a chave e o valor. No final do programa para cada um desses elementos imprima a frase “Uma das chaves é \*\*\*\* e o seu valor é \*\*\*\*”

**Missão 3**

Dado o seguinte hash:

Numbers = {a: 10, b: 30 2, c: 20, d: 25, e: 15}

Crie uma instrução que seleciona o maior valor deste hash e no final imprima a chave e valor do elemento resultante.