Aula 12

Hashes

Prof. Jalerson Lima

Apresentação

Nessa aula iremos estudar os *hashes*, que assim como os *arrays*, são estruturas de dados capazes de guardar um conjunto de dados, evitando a criação de várias variáveis. A principal diferença entre os *hashes* e os *arrays* é que os *hashes* indexam suas posições usando chaves, que podem ser *Strings* ou símbolos.

Objetivos

1. Compreender a utilidade e o funcionamento dos *hashes*;
2. Compreender a diferença entre *hashes* e *arrays*;
3. Aprender como criar, guardar e acessar elementos em *hashes*;
4. Aprender como iterar entre os elementos de um *hash*;
5. Exercitar os conhecimentos através de atividades práticas.

1. Introdução

Na última aula estudamos os *arrays*, que são importantes estruturas de dados que são capazes de guardar um conjunto de dados, evitando a criação de várias variáveis. Os *hashes* também são estruturas de dados, bem semelhantes aos *arrays*, que guardam um conjunto de dados. A principal diferença entre os *arrays* e os *hashes* é que, nos *arrays* cada posição é "endereçada" (indexada) por um número inteiro, enquanto nos *hashes*, cada posição é endereçada por uma chave.

Uma definição muito simples para os *hashes* é que eles são uma coleção de pares chave-valor. Confira a Figura 1, que ilustra a estrutura de um *hash*.

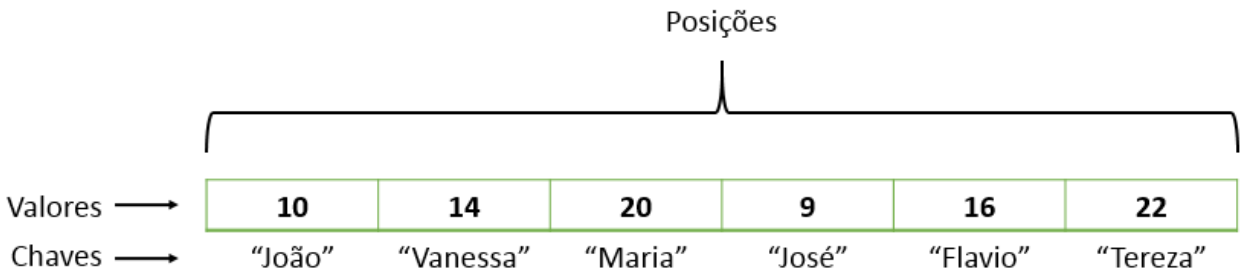

Figura 1 – Estrutura de um *hash*

Observe, na Figura 1, que a estrutura de um *hash* é muito semelhante à estrutura de um *array*, pois, conforme explicado anteriormente, ambas são estruturas que armazenam um conjunto de dados, evitando a criação de muitas variáveis. Observe também que cada posição do *hash* é marcado por uma chave, que, no caso do *hash* ilustrado na Figura 1, as chaves são *Strings*.

Contudo, as chaves dos *hashes* também podem ser qualquer tipo de dado, inclusive símbolos (*Symbol*). Um símbolo é um tipo de dado que representa apenas um marcador, uma etiqueta (tag), que não guarda valor algum. A única importância de um símbolo é marcar algo. Os símbolos são escritos com dois pontos na frente, e seus nomes obedecem às regras de nomenclatura de variáveis. O Exemplo de código 1 ilustra alguns símbolos.

```
:cor
:nome
:idade
:tipo
```

Exemplo de código 1 – Exemplos de símbolos

Portanto, conforme explicado anteriormente, os símbolos também podem ser usados como marcadores (chaves) de posições nos *hashes*.

2. Hashes em Ruby

Assim como os *arrays*, os *hashes* em Ruby também são dinâmicos, portanto eles irão aumentar de tamanho automaticamente quando forem necessárias novas posições para armazenar novos elementos. Outra semelhança com os *arrays* é que os *hashes* também são capazes de armazenar elementos de diferentes tipos de dados.

2.1 Criando um hash

Assim como fizemos com os *arrays*, vamos primeiramente aprender como criar um *hash*. Para isso, abra o IRB e digite o código ilustrado no Exemplo de código 2.

```
> meu_hash = Hash.new
=> {}
```

Exemplo de código 2 – Criando um *hash*

Digitando `meu_hash = Hash.new` (atenção para o H maiúsculo de `Hash`), estamos criando um novo *hash* vazio e guardando-o na variável `meu_hash`. O `{}` dado como resposta na linha 2 do Exemplo de código 2 indica que um *hash* vazio foi criado.

Uma forma mais simplificada para criar um *hash* vazio é ilustrado no Exemplo de código 3.

```
> meu_hash = {}
=> {}
```

Exemplo de código 3 – Criando um *hash*

Observe que simplesmente utilizando `{}` criamos um *hash* vazio. Portanto podemos criar *hashes* vazios utilizando `Hash.new` ou `{}`. Contudo, também é possível criar um *hash* com valores iniciais. Confira o Exemplo de código 4, que ilustra como criar um *hash* com valores inicias.

```
> meu_hash = { :nome => "Pedro", :idade => 28 }
=> {:nome => "Pedro", :idade => 28}
```

Exemplo de código 4 – Criando um *hash* com valores iniciais

Observe, no Exemplo de código 4, que a chave (que marca a posição) aparece antes de `=>`, enquanto o valor aparece depois. No exemplo acima usamos a chave `:nome` para marcar a posição do valor `"Pedro"` e a chave `:idade` para marcar a posição do valor `28`. A Figura 2 ilustra graficamente como ficou a estrutura do *hash* criado no Exemplo de código 4.

Figura 2 – Estrutura do *hash* criado no Exemplo de código 4

Atividade 12.1

Abra o IRB, crie um *hash* vazio e, em seguida, um *hash* com valores iniciais quaisquer.

2.2 Guardando valores no hash

Para adicionar novos elementos num *hash*, é preciso definir a posição na qual o elemento será guardado. Nos *hashes* essa posição é definida através de uma chave (*String* ou símbolo). Confira o Exemplo de código 5, que ilustra como adicionar novos elementos num *hash*.

```
> meu_hash[:altura] = 1.82
=> {:nome => "Pedro", :idade => 28, :altura => 1.82}
```

Exemplo de código 5 – Guardando valores num *hash*

Observe, no Exemplo de código 5, que usamos o nome da variável que guarda o *hash* (`meu_hash`), seguido da chave `:altura` (entre colchetes). A Figura 3 ilustra como ficou o *hash* após a adição do valor `1.82` na posição indexada pela chave `:altura`.

Figura 3 – Estrutura do *hash* após a adição do novo elemento

Atividade 12.2

Abra o IRB, crie um *hash* com valores iniciais e adicione um novo elemento qualquer.

2.3 Acessando valores do hash

Para acessar valores armazenados num *hash*, basta usar o nome da variável que guarda o *hash*, seguido da chave entre colchetes. Confira o Exemplo de código 6, que ilustra como acessar os valores de posições específicas de um *hash*.

```
> meu_hash[:nome]
=> "Pedro"
> meu_hash[:idade]
=> 28
> meu_hash[:altura]
=> 1.82
> meu_hash[:uma_chave_inexistente]
=> nil
```

Exemplo de código 6 – Acessando valores do *hash*

Observe, no Exemplo de código 6, que quando tentamos acessar uma posição usando uma chave qualquer inexistente, o valor retornado é `nil`, pois não existe valor algum armazenado nessa posição.

Atividade 12.3

Abra o IRB, crie um *hash* com valores iniciais e depois acesse um desses valores, conforme ilustrado no Exemplo de código 6.

2.4 Métodos e operadores auxiliares

Assim como nos *arrays*, existem alguns métodos e operadores que auxiliam a manipulação de dados em *hashes*. Assim como foi feito na aula passada, iremos apresentar apenas alguns desses métodos e operadores, contudo você pode conferir um detalhamento de todos eles na página oficial da documentação do Ruby: http://docs.ruby-lang.org/en/2.0.0/Hash.html (em inglês).

Para ilustrar os métodos e operadores auxiliares, iremos utilizar os dois *hashes* apresentados no Exemplo de código 7.

```
> hash1 = { :um => 1, :dois => 2, :tres => 3 }
=> {:um => 1, :dois => 2, :tres => 3}
> hash2 = { :tres => 3, :quatro => 4, :cinco => 5 }
=> {:tres => 3, :quatro => 4, :cinco => 5}
```

Exemplo de código 7 – *Hashes* que serão usados como exemplo

O operador `==`, ilustrado no Exemplo de código 8, verifica se os dois *hashes* são iguais. Eles são considerados iguais quando eles têm a mesma quantidade de pares chave-valor, e se todos os pares chave-valor são iguais entre os *hashes*.

```
> hash1 == hash2
=> false
```

Exemplo de código 8 – O operador ==

O método `clear`, ilustrado no Exemplo de código 9, apaga todos os pares chave-valor armazenados no *hash*.

```
> hash1.clear
=> {}
```

Exemplo de código 9 – O método *clear*

O método `delete`, ilustrado no Exemplo de código 10, recebe uma chave como parâmetro e apaga o par chave-valor correspondente.

```
> hash2.delete(:tres)
=> 3
```

Exemplo de código 10 – O método *delete*

O método `empty?`, ilustrado no Exemplo de código 11, retorna verdadeiro caso o *hash* esteja vazio e falso caso contrário.

```
> hash1.empty?
=> false
```

Exemplo de código 11 – O método *empty?*

Os métodos `has_key?`, `include?`, `key?` e `member?`, ilustrados no Exemplo de código 12, recebem uma chave como parâmetro, retornam verdadeiro caso a chave já esteja presente no *hash* e falso caso contrário.

```
> hash2.has_key?(:quatro)
=> true
> hash2.include?(:dez)
=> false
> hash1.key?(:um)
=> true
> hash1.member?(:dois)
=> true
```

Exemplo de código 12 – Os métodos *has_key?*, *include?*, *key?* e *member?*

Os métodos `has_value?` e `value?`, ilustrados no Exemplo de código 13, recebem um valor como parâmetro, retornam verdadeiro caso esse valor esteja armazenado no *hash* e falso caso contrário.

```
> hash1.has_value?(2)
=> true
> hash2.value?(10)
=> false
```

Exemplo de código 13 – Os métodos *has_value?* e *value?*

O método `keys`, ilustrado no Exemplo de código 14, retorna um *array* com as chaves presentes no *hash*.

```
> hash2.keys
=> [:tres, :quatro, :cinco]
```

Exemplo de código 14 – O método *keys*

Os métodos `size` e `length`, ilustrados no Exemplo de código 15, retornam a quantidade de elementos armazenados no *hash*.

```
> hash1.size
=> 3
> hash2.length
=> 3
```

Exemplo de código 15 – Os métodos *size* e *length*

Atividade 12.4

Abra o IRB, escolha e use três funções e/ou operadores auxiliares para manipular *hashes*.

2.5 Iterando em hashes

Assim como nos *arrays*, em diversas situações também será necessário iterar, ou seja, percorrer entre os elementos armazenados num *hash*, realizando alguma verificação ou processamento com cada um desses elementos.

Existem várias maneiras de iterar entre os elementos de um *hash*. O Exemplo de código 16 ilustra como iterar usando o *each*. Observe, na linha 3, que especificamos os nomes das variáveis (`chave` e `valor`) que irão armazenar a chave e o valor de cada posição do *hash*. Assim como nos *arrays*, a cada iteração do laço `each`, as variáveis `chave` e `valor` irão assumir uma chave e um valor de uma posição do *hash*.

```
1  meu_hash = { :um => 1, :dois => 2, :tres => 3 }
2
3  meu_hash.each do |chave, valor|
4    puts "A posição #{chave} guarda o valor #{valor}"
5  end
```

Exemplo de código 16 – Iterando entre os elementos com *each*

O Exemplo de código 17 ilustra outra forma de iterar entre os elementos de um *hash*. Nesse exemplo também usamos o `each`, contudo, usamos as chaves (`{` e `}`) ao invés de usar o `do` e o `end` para marcar o início e o fim do bloco de código.

```
1  meu_hash = { :um => 1, :dois => 2, :tres => 3 }
2
3  meu_hash.each { |chave, valor|
4    puts "A posição #{chave} guarda o valor #{valor}"
5  }
```

Exemplo de código 17 – Iterando entre os elementos com *each*

O Exemplo de código 18 ilustra outra forma de iterar entre os elementos de um *hash*, mas dessa vez utilizando o `each_pair`, que funciona da mesma forma que os exemplos apresentados anteriormente.

```
1  meu_hash = { :um => 1, :dois => 2, :tres => 3 }
2
3  meu_hash.each_pair do |chave, valor|
4    puts "A posição #{chave} guarda o valor #{valor}"
5  end
```

Exemplo de código 18 – Iterando entre os elementos com *each_pair*

Em determinadas situações pode ser necessário iterar apenas entre as chaves, e não entre os valores. Para esses casos existe o `each_key`, ilustrado no Exemplo de código 19.

```
1  meu_hash = { :um => 1, :dois => 2, :tres => 3 }
2
3  meu_hash.each_key do |chave|
4    puts "A posição #{chave} guarda o valor #{meu_hash[chave]}"
5  end
```

Exemplo de código 19 – Iterando entre os elementos com *each_key*

Conforme ilustrado no Exemplo de código 20, o `each_key` também pode ser escrito usando chaves (`{` e `}`) ao invés de `do` e `end`.

```
1  meu_hash = { :um => 1, :dois => 2, :tres => 3 }
2
3  meu_hash.each_key { |chave|
4    puts "A posição #{chave} guarda o valor #{meu_hash[chave]}"
5  }
```

Exemplo de código 20 – Iterando entre os elementos com *each_key*

Também é possível utilizar o `for` para iterar entre os elementos de um *hash*, conforme ilustrado no Exemplo de código 21. Observe que, após o `for`, é necessário especificar os nomes das variáveis que irão guardar as chaves e os valores do *hash* (`chave` e `valor`). O nome da variável que armazena o *hash* é especificado após o `in`.

```
1  meu_hash = { :um => 1, :dois => 2, :tres => 3 }
2
3  for chave, valor in meu_hash
4    puts "A posição #{chave} guarda o valor #{valor}"
5  end
```

Exemplo de código 21 – Iterando entre os elementos com *for*

Atividade 12.5

Crie um *script* em Ruby. Nesse *script*, crie um *hash* com valores iniciais, escolha e use dois laços para iterar entre os elementos desse *hash*. Mostre na tela cada chave e valor armazenado no *hash*.

Atividade 12.6

Crie um *script* em Ruby que leia o nome e a idade de várias pessoas. O *script* deve parar de ler nomes e idades quando o usuário teclar ENTER sem digitar nada para nome e idade. Por fim, o *script* deve calcular e mostrar:

- O nome e a idade da pessoa mais idosa;
- O nome e a idade da pessoa mais jovem;
- A quantidade de pessoas maiores de idade (18 anos ou mais);
- A quantidade de pessoas menores de idade (menos de 18 anos);
- A média das idades.

Resumindo

Essa aula apresentou a utilidade e como funcionam os *hashes* em Ruby. Aprendemos como criar *hashes*, guardar e ler elementos armazenados, utilizar métodos e operadores para manipulá-los, e exercitamos nossos conhecimentos através de atividades práticas.

Referências

COMMUNITY, R. Class Hash. **Ruby Documentation**, 2015. Disponivel em: <http://docs.ruby-lang.org/en/2.0.0/Hash.html>. Acesso em: 10 fev. 2015.

POINT, T. Ruby Tutorial. **Tutorials Point**, 2015. Disponivel em: <http://www.tutorialspoint.com/ruby/>. Acesso em: 12 nov. 2015.

RANGEL, E. **Conhecendo Ruby**. [S.l.]: Leanpub, 2014.

SOUZA, L. **Ruby - Aprenda a programar na linguagem mais divertida**. 1ª. ed. São Paulo: Casa do Código, v. I, 2012.