

Lógica de Programação

Aula 3 - Arrays, hashes e funções

Arrays

- ▶ Já sabemos como atribuir variáveis no Ruby
 - ▶ nome = “João” (string)
 - ▶ cidade = “Maceió” (string)
 - ▶ idade = 20 (fixnum)
 - ▶ nota = 8.5 (float)
- ▶ Mas e se tivermos vários nomes, cidades, idades e notas no nosso programa?
 - ▶ nome1 = “João”, nome2 = “José”, nome3 = “Maria”
 - ▶ cidade1 = “Maceió”, cidade2 = “Arapiraca”,
 - ▶ cidade3 = “Marechal Deodoro”

Arrays

Como podemos melhorar?

Arrays!

Coleções de elementos agrupados em uma mesma variável, com seus valores separados por índices numéricos, onde:

- ▶ Primeiro índice: 0
- ▶ último índice: $n - 1$ (n = total de elementos do array)

Arrays

Ficou confuso? Calma...

- Array com vários nomes:

Array de nomes com 5 elementos. $n = 5$

- ▶ `nomes = ["João", "José", "Maria", "Joaquina", "Austragísio"]`
 - ▶ `puts nomes[0]` # Primeiro índice: 0. João
 - ▶ `puts nomes[1]` # José
 - ▶ `puts nomes[2]` # Maria
 - ▶ `puts nomes[3]` # Joaquina
 - ▶ `puts nomes[4]` # Último índice: $n - 1$. Austragísio

Arrays

Vários tipos de elementos podem estar em um mesmo array

- ▶ Ruby: dinamicamente tipada!

Exemplo:

- ★ dados = ["João", 20, "Maceió", 8.5]
 - ★ puts "#{dados[0]} tem #{dados[1]} anos, mora em #{dados[2]} e tirou
 - ★ "#{dados[3]} na prova."
-
- ▶ Você precisa apenas estruturar seu programa para isso
 - ▶ Criar uma implementação que se adeque a tratar um array com essa estrutura.
 - ▶ Lembre-se: você, programador(a), é o dono do algoritmo. Você dita as regras aqui!

Operando sobre arrays

- ▶ Iniciando um array vazio
 - ▶ `array = Array.new`
 - ▶ `array = []`
- ▶ Total de elementos no array
 - ▶ `array.length`
- ▶ Adicionando elementos durante execução
 - ▶ `array << "elemento"`
 - ▶ `array << 199`
- ▶ Acessando o primeiro e o último elemento
 - ▶ `array.first`
 - ▶ `array.last`
- ▶ Acessando o elemento na posição n
 - ▶ `array[n]`
 - ▶ `array.at(n)`

▶ <https://ruby-doc.org/core-2.4.1/Array.html>

Iterando sobre arrays

- ▶ Como fazemos para percorrer um array?
 - ▶ Liste todos os elementos do array nomes:

```
nomes = ["João", "José", "Maria", "Joaquim", "Austragísio"]
```

- ▶ Estrutura for

```
for nome in nomes  
  puts nome  
end
```

- ▶ Estrutura each

```
nomes.each do |nome|  
  puts nome  
end
```

```
for i in 0 .. nomes.length-1  
  puts nomes[i]  
end
```

```
nomes.each_with_index do |nome, indice|  
  puts nomes[indice]  
end
```

Combinando arrays

- ▶ Podemos ter vários arrays para guardar diferentes informações, e combiná-las a nosso gosto
 - ▶ Lembre-se: você diz ao seu programa o que e como ele deve fazer
- ▶ Imagine:

```
marcas = ["Ford", "Fiat", "Volkswagen", "Chevrolet"]
```

```
modelos = ["Fiesta", "Uno", "Gol", "Corsa"]
```

```
marcas.each_with_index do |marca, indice|
```

```
  modelo = modelos[indice]
```

```
  puts "O modelo mais popular da #{marca} é o #{modelo}"
```

```
end
```

- ▶ O Modelo mais popular da ford é o fiesta
- ▶ O Modelo mais popular da fiat é o uno
- ▶ O Modelo mais popular da volkswagen é o gol
- ▶ O Modelo mais popular da chevrolet é o corsa

Hashes

- ▶ Combinar arrays pode ser até útil, mas não muito prático.
 - ▶ E se pudéssemos informar, em uma mesma variável, todas as marcas e seus modelos populares?
- ▶ Hashes!
 - ▶ Coleções de elementos que, como o array, se organizam em índices e valores...
 - ▶ Mas os índices não precisam ser sempre numéricos!
 - ▶ Nós mesmos podemos criar os índices.

Hashes

- ▶ Criando um hash:
 - ▶ `hash = Hash.new`
 - ▶ `hash = {}` # chaves no lugar de colchetes
- ▶ Adicionando e acessando elementos
 - ▶ `hash["chave"] = "valor"`
 - ▶ `puts hash["chave"]` # Imprimirá "valor"
- ▶ Chaves como símbolos
 - ▶ `hash[:chave] = "valor"`
 - ▶ `hash = {chave: "valor"}`
 - ▶ `hash = {:chave => "valor"}`
 - ▶ `puts hash[:chave]`
- ▶ Todos os valores de um Hash (método values)
 - ▶ `hash = {nome: "José", idade: 25, cidade: "Maceió"}`
 - ▶ `puts hash.values`
 - ▶ # Imprimirá os valores como array ["José", 25, "Maceió"]
 - ▶ <https://ruby-doc.org/core-2.4.1/Hash.html>

Hashes

Hash dos carros populares por marca

```
carros = {  
    ford: ["Fiesta", "Focus"],  
    fiat: ["Uno", "Palio"],  
    chevrolet: ["Corsa", "Celta"],  
    volswagen: "Gol"  
}  
  
puts carros[:ford]  
  
# Fiesta  
  
# Focus
```

Exercicios

1. Crie um programa que calcule a média de um array com 10 números.
2. Crie um programa que receba vários nomes e adicione a uma lista de convidados. Quando o usuário digitar “FIM”, imprima todos os nomes da lista na tela.
3. Crie um programa que utilize um hash com as chaves S, SU, N, NE, C (respectivamente: Sul, Sudeste, Norte, Nordeste e Centro-Oeste) e seus respectivos estados como valores (não precisa escrever todos, mas pelo menos um de cada região!). O usuário digitará uma das chaves e o programa imprimirá os estados correspondentes.

Exiba uma mensagem caso o valor digitado pelo usuário não seja uma chave do hash criado!

Funções

Chegamos até aqui fazendo vários programas legais:

- ▶ Imprimimos nossos nomes na tela
- ▶ Fizemos operações matemáticas
- ▶ Brincamos com entradas do teclado do usuário
- ▶ Criamos regras (se, senão se, senão, enquanto, até que...)

Perceba que, a cada exercício, nós criamos um arquivo, com suas devidas variáveis e os comandos necessários para executar o que queríamos!

Funções

E se quisermos executar o mesmo comando mais de uma vez em diferentes pontos do programa?

Funções: blocos de códigos que podem ser reutilizados, através de sua chamada, em qualquer ponto do programa.

```
def ola_mundo  
    puts "Olá, mundo!"  
end
```

Funções

Estrutura de uma função:

- ▶ `def` é a palavra reservada do Ruby que é utilizada para dizer que você está definindo uma função.
- ▶ `ola_mundo` é o identificador da sua função. Uma vez definido com esse identificador, seu bloco de códigos será executado sempre que o identificador for encontrado.
- ▶ `end` determina o final da definição da função.

```
def ola_mundo  
  puts "Olá, mundo!"  
end
```

Funções

- ▶ Retorno de um método
 - ▶ A última linha de código que define o método (antes do end) é o seu retorno. É aquilo que o método vai retornar para quem o chamar tratar da forma que desejar.

```
def cem_ao_quadrado
```

```
    100 * 100
```

```
end
```

```
cem_ao_quadrado # Executa o método mas não faz nada
```

```
puts cem_ao_quadrado # Executa o método e imprime seu RETORNO
```


Funções

- ▶ Parâmetros

- ▶ São variáveis que passamos para as nossas funções, de forma que os códigos contidos nelas possam trabalhar de acordo com os valores que atribuímos às mesmas.

```
def numero_ao_quadrado(numero)
```

```
    numero * numero
```

```
end
```

```
puts numero_ao_quadrado(4) # 16
```

```
def ola_fulano(nome)
```

```
    "Ola #{nome}"
```

```
end
```

```
puts ola_fulano "Ernande" # Ola Ernande
```

Funções

- ▶ Parâmetros
 - ▶ Pode haver mais de um.
 - ▶ Experimente: hashes!

Também recebem valores default

```
def numero_ao_quadrado(numero = 100)
```

```
  numero * numero
```

```
end
```

```
def ola_fulano(nome, cidade = "Maceió")
```

```
  "Ola #{nome} de #{cidade}"
```

```
end
```

Exercícios - Funções

1. Crie uma função que receberá como parâmetro a hora do dia (valor de 0 à 23) e dirá se é manhã, tarde, noite ou madrugada.

Madrugada = para horas entre 0 à 5.

Dia = para horas entre 6 à 12.

Tarde = para horas entre 13 à 17

Noite = para horas entre 18 à 23

Exercícios - Funções

2. Um salgado na cantina da Conhecimento Digital custa R\$ 4,00. A partir de 5 salgados, você tem desconto de 5%. De 10 em diante, 10%. Crie um programa com duas funções:

- ▶ Uma para calcular o preço de acordo com a quantidade de salgados comprada.
- ▶ Uma para calcular o desconto de acordo com a quantidade de salgados comprada.

Mostre na tela o valor total da compra, já com os descontos (se houver algum...).

Apêndice - Enumerable

Módulo para manipular coleções

Trata-se de um módulo que, em linhas de código menores e “mais limpas”, faz operações que costumamos fazer com os loops em arrays e hashes.

<https://ruby-doc.org/core-2.4.1/Enumerable.html>

Exemplo:

Transforme todos os números do array em números negativos

Apêndice - Enumerable

Solução com for:

```
numeros = [1,2,3,4,5,6]
for i in 0 .. numeros.length-1
  numeros[i] = -numeros[i]
end
puts numeros # -1, -2, -3 ...
```

Apêndice - Enumerable

Solução com método map:

```
numeros = [1,2,3,4,5,6]
```

```
negativos = numeros.map{|x| -x}
```

```
puts negativos # -1, -2, -3 ...
```

Apêndice - Enumerable

Outros métodos interessantes:

- all?
- any?
- take
- sort
- sort_by
- select
- delete_if