kazap Academy

Ruby - v3.x

Professores:

Bruno Faboci

Rodrigo Paschoaletto

Monitores:

Paulo Otilio

Pedro Dalben

Academy

3rd Edition

kazap Academy

Cronograma

Ruby Básico

- IRB;
- Declaração de variáveis;
- Tipos primitivos;
- Entrada e saída padrão;
- Comentários;
- Strings e Interpolação de variáveis;
- Coerção (Casting);
- Operadores Aritméticos;
- Operadores Relacionais;
- Operadores de Atribuição;
- Operadores Lógicos;
- Criando Métodos;
- Estruturas Condicionais;
- Range e Estruturas de Repetição;
- Hash;
- Array.

Hello, World!

kazap Academy

IRB - Interactive Ruby Shell

O IRB é um ambiente de programação interativo, que recebe entradas do usuário, as processa e retorna o resultado.

É iniciado a partir do comando *irb* no terminal.

Declaração de Variáveis

kazap Academy

- Variáveis Locais
  - São as mais comuns quando utilizamos o Ruby. Variáveis locais só podem ser usadas dentro do escopo em que foram criadas.
  - *nome = 'João'*
- Constantes
  - Constantes declaradas no formato CamelCase e são utilizadas para armazenar valores que não queremos que sofram alteração. Por convenção, declaramos constantes no formato UPCASE.
  - *Nome = 'João'*
  - *NOME = 'João'*
- Variáveis Globais
  - São compartilhadas entre todas as instâncias de uma classe.
  - *$nome = 'João'*
- Variáveis de Instância
  - São restritas a cada instância de uma classe.
  - *@nome = 'João'*

Tipos Primitivos

kazap Academy

- Integer
  - 1; 548; -15
- Float
  - 2.5; 150.8; -15.4
- String
  - “Hello”; “string”; “TESTE”
- Boolean
  - true / false

Verificar o tipo da variável

kazap Academy

Podemos verificar qual o tipo primitivo de uma variável utilizando os comandos abaixo:

- *object.class*
  - *retorna a classe do objeto.*
- *object.is_a?(String)*
  - *retorna true se a classe passada for a classe do objeto.*

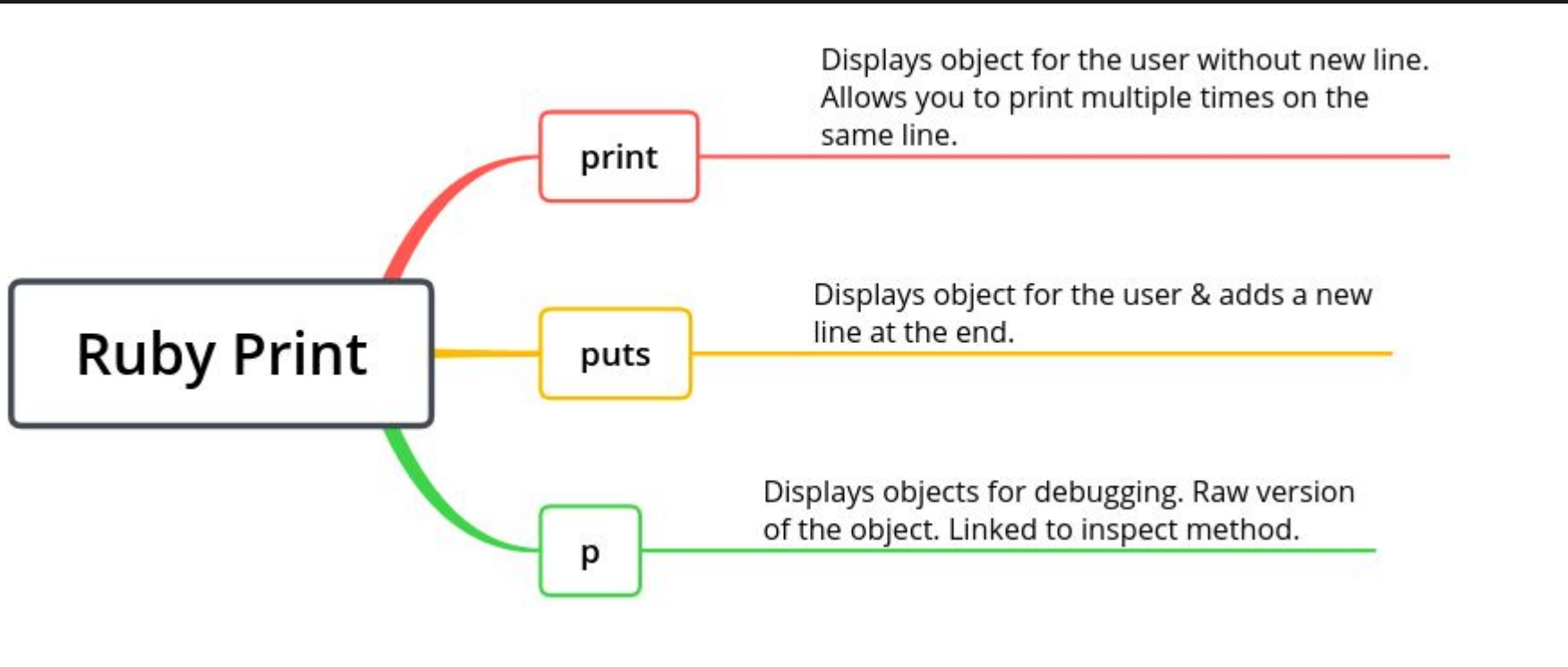
Entrada e Saída Padrão

kazap Academy

- *puts*
  - exibe o conteúdo adicionando quebra de linha
- *print*
  - exibe o conteúdo sem adicionar quebra de linha
- *p*
  - exibe a versão mais “raw” (crua) do objeto
- *gets*
  - recebe dados do usuário
- *gets.chomp*
  - recebe os dados e remove a quebra de linha

Entrada e Saída Padrão

kazap Academy



Comentários

kazap Academy

- Uma linha
  - #
- Múltiplas linhas

```
=begin

xxxxxxxxx

=end
```

Strings e interpolação de variáveis

kazap Academy

- Interpolação
  - #{ }
- Métodos (destrutivos e não-destrutivos)
  - capitalize | capitalize!
  - downcase | downcase!
  - upcase | upcase!
  - strip
  - reverse
  - length / size
  - empty?

Coerção (Casting)

kazap Academy

O Ruby nos permite tipificar explicitamente os tipos de dados por meio da coerção ou casting. Alguns exemplos:

- *.to_i*
  - to integer
- *.to_f*
  - to float
- *.to_s*
  - to string
- *.to_sym*
  - to symbol

Operadores Aritméticos

kazap Academy

- adição
  - +
- subtração
  - -
- multiplicação
  - *
- divisão
  - /
- módulo
  - %
- exponenciação
  - **

Operadores Relacionais

kazap Academy

- menor
  - <
- maior
  - >
- menor ou igual
  - <=
- maior ou igual
  - >=
- igual
  - ==
- diferente
  - !=

Operadores Atribuição

kazap Academy

- =
  - *x = 2*
- +=
  - *x += y*
  - equivalente a “ x = x + y “
- -=
  - *x -= y*
  - equivalente a “ x = x - y “
- *=
  - *x *= y*
  - equivalente a “ x = x * y “
- /=
  - *x /= y*
  - equivalente a “ x = x / y “
- %=
  - *x %= y*
  - equivalente a “ x = x % y “
- **=
  - *x **= y*
  - equivalente a “ x = x ** y “

Operadores Lógicos

kazap Academy

x = 5

y = 4

- && => “e”
  - x == 5 && y == 4
    - >>>> true
- || => “ou”
  - x == 5 || y == 5
    - >>>> true
- ! => “negação”
  - ! (x == 5 && y == 4)
    - >>>> false

Criando Métodos

kazap Academy

Em Ruby, os métodos são equivalentes às funções de outras linguagens.

Os nomes dos métodos devem começar com letra minúscula e usar o underscore para separar palavras, caso o nome seja composto.

A sintaxe dos métodos é bastante simples:

```
def nome_do_metodo(argumentos)
    #alguma lógica
end
```

Criando Métodos

kazap Academy

Em métodos, os argumentos são opcionais. Podemos criar métodos que não recebem argumentos.

```
def dizer_oi
    "Oi"
end
```

Métodos que recebem argumentos, geralmente tem essa cara:

```
def dizer_oi(nome)
    "Oi #{nome}"
end
```

```
def soma(num1, num2)
    num1 + num2
end
```

Criando Métodos

kazap Academy

É possível passar valores padrão para os argumentos.

```
def dizer_oi(nome = “Fulano”)
    “Oi #{nome}”
end
```

Criando Métodos

kazap Academy

Retornando Valores de um Método

Todo método em ruby, por padrão, retorna o valor da última instrução. Por exemplo:

```
def test
    i = 50
    j = 100
    k = 200
end
```

Este método (à esquerda), quando for chamado, irá retornar a última variável declarada, *k.*

Para retornar um ou mais valores, utilizamos o *return.*

```
def test
    i = 50
    j = 100
    k = 200
    return i, j, k
end
```

Estruturas Condicionais

kazap Academy

- *if … elsif … else*
- *unless*
- *case … when*

Exemplo:

```
num = 8
if num.zero?
        puts “Valor inválido!”
elsif (num % 2).zero?
        puts “O número #{num} é par”
else
        puts “O número #{num} é ímpar”
end
```

Ranges e Estruturas de Repetição

kazap Academy

Range

Um range representa um intervalo de valores com início e fim. Pode ser criado usando os literais ***s..e*** e ***s…e*** .

Quando construído usando ***s..e*** o valor final será incluído.

Quando construído usando ***s...e*** o valor final será excluído.

Exemplos:

(1..10).to_a >>>> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] → *último valor incluído*

(1…10).to_a >>>> [1, 2, 3, 4, 5, 6, 7, 8, 9] → *último valor excluído*

Ranges e Estruturas de Repetição

kazap Academy

- For

```
for variável in expressão do
    #código
end
```

- While

```
while condição do
    #código
end
```

- Until

```
until condição do
    #código
end
```

- Each

```
itens.each do |item|
    #código
end
```

Símbolos (Symbols)

kazap Academy

Símbolos Ruby podem ser definidos como *“objetos de valor escalar usados como identificadores, mapeando strings imutáveis para valores internos fixos”*.

Na prática, isso significa que símbolos são strings imutáveis, ou seja, um símbolo permanecerá o mesmo até ser destruído. Exemplo:

Strings

“Hello”.object_id >>>> 225920

“Hello”.object_id >>>> 228960

“Hello”.object_id >>>> 232000

“Hello”.object_id >>>> 236120

Symbols

:Hello.object_id >>>> 3062748

:Hello.object_id >>>> 3062748

:Hello.object_id >>>> 3062748

:Hello.object_id >>>> 3062748

Hash

kazap Academy

Hash é uma coleção de pares “chave-valor” (key-value). Pode ser criado usando o método ***Hash.new*** ou a forma literal ***hash = { }.*** Exemplos:

hash = { a : 1, b : “2” } >>>> { :a => 1, :b=>”2” }

pessoa = { nome: “Chicó”, idade: 28 } >>>> { :nome => “Chicó, :idade=>28 }

pessoa = Hash.new >>>> { }

pessoa.merge!(nome: “Chicó”) >>>> { nome => “Chicó }

pessoa[:idade] = 28 >>>> { nome => “Chicó, :idade=>28 }

Hash

kazap Academy

Para acessar os elementos contidos no *hash,* utilizamos sua chave.

```
pessoa[:nome]  >>>  “Chicó”

pessoa[:idade]  >>>  28
```

É possível alterar o valor das chave:

```
pessoa[:nome] = “Outro Nome”
```

E deletar valores:

```
pessoa.delete(:idade)
```

Hash - Métodos

kazap Academy

- any?
- empty?
- has_value? | value?
- include?
- length | size

Hash - Iteração

kazap Academy

Ruby nos fornece algumas formas bastante úteis de iterar sobre hashs. Algumas delas são *each*, *each_pair*, *each_key* e *each_value*. Exemplos:

```
hash = { a: 1, b: 2 }  >>>>  { :a => 1, :b => 2 }
```

```
#each
hash.each do |key, value|
        puts "#{key} - #{value}"
end
```

```
#each_pair
hash.each_pair do |pair|
        puts pair.to_s
end
```

```
#each_key
hash.each_key do |key|
        puts key.to_s
end
```

```
#each_value
hash.each_value do |value|
        puts value.to_s
end
```

Hash - Iteração

kazap Academy

Ruby também nos permite criar hashs aninhados (nasted hashs) e também podemos iterar sobre eles.

```
hash = { a: { x: 1, y: 2 }, b: { x: 3, y: 4 } }

>>>>  { :a=>{ :x=>1, :y=>2 }, :b=>{ :x=>3, :y=>4 } }

#each
hash.each do |key, value|
        puts key.to_s
        value.each do |key2, value2|
                puts “#{key2} - #{value2}”
        end
end
```

Arrays

kazap Academy

Array é uma coleção ordenada de objetos. Pode ser criado usando o método “***Array.new***” ou utilizando a forma literal “***array = [ ]*** ”. Exemplos:

array = Array.new >>>> [ ]

array = Array.new(3) >>>> [nil, nil, nil]

array = Array.new(3, “blá”) >>>> [“blá”, “blá”, “blá”]

array = [1, 2, 3, 4, 5] >>>> [1, 2, 3, 4, 5]

Arrays

kazap Academy

Acessando Elementos

Podemos acessar os elementos de um Array utilizando o método array#[ ].

Esse método pode receber um único argumento (valor do índice), um par de argumentos (start e length) ou um range. Índices negativos começam a contar a partir do final do array.

Os índices sempre começam em zero (0). Exemplos:

```
ary = [1, 2, 3, 4, 5]
ary[2]  >>>>  3
ary[1, 3]  >>>>  [2, 3, 4]
ary[0..2]  >>>>  [1, 2, 3]
```

```
ary[-1]  >>>>  5
```

Arrays

Acessando Elementos

kazap Academy

Há outros métodos que podem ser usados para acessar os elementos de um array. Alguns deles são ***at, fetch, first, last, taken*** e ***drop***.

O método ***at***, retorna o elemento na posição (índice) informado. Exemplo:

```
ary = [1, 2, 3, 4, 5]

ary.at(2)  >>>>  3
```

O ***fetch*** funciona de forma análoga ao ***at***, no entanto, nos permite criar um tratamento de erro para quando o valor informado não se encontrar dentro do limite do array. Exemplo:

```
ary = [1, 2, 3, 4, 5]

ary.fetch(50)  >>>>  index 50 outside of array bounds: -5...5 (IndexError)

ary.fetch(50, “Errrou”)   >>>>  Errrou
```

Arrays

Acessando Elementos

kazap Academy

Os métodos ***first*** e ***last*** retornam, respectivamente, o primeiro e último elemento do array. Exemplo:

```
ary = [1, 2, 3, 4, 5]

ary.first >>>> 1   |   ary.last >>>> 5
```

O método ***take***, retorna os primeiros *n* elementos do array. Exemplo:

```
ary.take(3) >>>> [1, 2, 3]
```

E o método ***drop*** é similar ao ***take***, no entanto, retorna os elementos que estão após os *n* elementos do array. Exemplo:

```
ary.drop(3) >>>> [4, 5]
```

Arrays

Obtendo informações

kazap Academy

Saber a quantidade de elementos presentes em um array pode ser uma informação importante. Para obter essa informação, podemos usar os métodos ***lenght***, ***count*** ou ***size***.

```
ary = [1, 2, 3, 4, 5]

ary.length >>>> 5 | ary.count >>>> 5 | ary.size >>>> 5
```

Para verificar se há elementos presentes em um array, podemos utilizar os métodos ***any?*** e ***empty?***

```
ary.any? >>>> true | ary.empty? >>>> false
```

E para verificar se existe um valor específico no array, podemos utilizar o método ***include?***

```
ary.include?(3) >>>> true | ary.include?(8) >>>> false
```

Arrays

kazap Academy

Adicionando Itens

A inserção de elementos em um array pode ser feita por meio de três comandos: ***push*** ou **<<**, ***unshift*** e ***insert***.

o comando ***push*** ou **<<**, irá inserir o novo elemento no fim do array. O ***unshift*** fará a inserção do novo elemento no início do array. Já o comando ***insert*** nos permite dizer em qual posição (índice) queremos inserir o novo elemento. Exemplos:

```
lista = [1, 2, 3]  >>>>  [1, 2, 3]

lista.push(4)  >>>>  [1, 2, 3, 4]     |     lista << 4  >>>>  [1, 2, 3, 4]

lista.unshift(0)  >>>>  [0, 1, 2, 3, 4]

lista.insert(2, “dois”)  >>>>  [0, 1, “dois”, 2, 3, 4]
```

Arrays

kazap Academy

Removendo Itens

Podemos remover elementos de um array utilizando os comandos ***pop***, ***shift***, ***delete_at*** e ***delete***.

O comando ***pop*** remove o último elemento do array. O ***shift*** remove o primeiro elemento. ***delete_at*** remove o elemento do índice informado, e ***delete*** remove do array todos os elementos informados como parâmetro. Exemplos:

```
lista = [1, 2, 2, 3, 4, 5]  >>>>  [1, 2, 2, 3, 4, 5]

lista.pop  >>>>  [1, 2, 2, 3, 4]

lista.shift  >>>>  [2, 2, 3, 4]

lista.delete_at(2)  >>>>  [2, 2, 4]

lista.delete(2)  >>>>  [4]
```

Arrays

Removendo Itens

kazap Academy

Além dos métodos já citados, existem outros dois métodos que são muito úteis para remover elementos mais específicos de arrays. São eles o ***compact*** e ***uniq***.

O ***compact*** remove valores nulos (nil) do array e o ***uniq*** remove valores duplicados.

Exemplos:

```
lista = [1, 2, nil, 4, 5, nil ]  >>>>  [1, 2, nil, 4, 5, nil]

lista.compact!  >>>>  [1, 2, 4, 5]


lista = [1, 2, 2, 3, 4, 5, 5, 6 ]  >>>>  [1, 2, 2, 3, 4, 5, 5, 6]

lista.uniq! >>>>  [1, 2, 3, 4, 5, 6]
```

Arrays

Arrays aninhados | Nested Arrays

kazap Academy

Arrays podem conter todo tipo de dados, incluindo outros arrays.

Exemplos:

```
lista = [ [ 1, 2 ], [ 3, 4 ] ]  >>>>  [ [1, 2 ], [ 3, 4] ]
```

Podemos acessar os elementos de um array aninhado da mesma forma como acessamos elementos em um array comum.

```
lista[0] >>>>  [1, 2]

lista[0][0] >>>>  [1]
```

E também adicionar elementos:

```
lista[1] << 5   |   lista[1].push(5)

>>>>  [ [1, 2 ], [ 3, 4, 5] ]
```

Arrays

Arrays aninhados | Nested Arrays

kazap Academy

De forma análoga ao Hash, também podemos iterar sobre arrays aninhados. A diferença aqui é que a iteração ocorre em níveis. Exemplo:

```
array = [ [ 1, 2 ], [ 3, 4 ] ]  >>>>  [ [1, 2 ], [ 3, 4] ]

    array.each do |x|
      x.each do |y|
          puts y.to_s
      end
    end
```

Arrays

Select

kazap Academy

O método ***select*** funciona como um filtro. Para cada elemento do array, ele aplica a condição passada no bloco de código e retorna apenas os elementos que atendem a essa condição. Exemplo:

```
numeros = [1, 2, 3, 4, 5]
numeros.select { |numero| numero > 2 }  >>>>  [3, 4, 5]
```

Outro exemplo:

```
numeros.select { |numero| numero.even? }  >>>>  [2, 4]
```

Arrays

Reduce

kazap Academy

O método ***reduce*** combina todos os elementos contidos no array, aplicando uma operação especificada por um símbolo, e retorna o resultado dessa combinação.
Exemplo:

```
numeros = [1, 2, 3, 4, 5]
numeros.reduce( :+ )  >>>>  15
numeros.reduce( :* )  >>>>  120

letras = [“a”, “b”, “c”, “d”, “e”]
letras.reduce( :+ )  >>>>  “abcde”
```

Arrays

Map

kazap Academy

O ***map*** é um método que “mapeia” cada elemento do array e aplica a cada um uma condição ou lógica definida dentro do bloco de código. Exemplos:

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

numeros.map { |numero| numero * 2 }  >>>>  [2, 4, 6, 8, 10]


nomes = [“joao”, “maria”, “chico” ]

nomes.map { |nome| nome.capitalize }  >>>>  [“Joao”, “Maria”, “Chico”]
```

Arrays

Map

kazap Academy

Um exemplo mais complexo:

```
numeros = [1, 2, 3, 4, 5]
def potenciacao(num)
    num ** 2
end

numeros.map {  |num| potenciacao(num)  } >>>>  [1, 4, 9, 16, 25]
```

Referências Bibliográficas

kazap Academy

- IRB
  - https://docs.ruby-lang.org/en/3.0/IRB.html
  - https://en.wikipedia.org/wiki/Interactive_Ruby_Shell
  - https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop
- Variáveis
  - https://blog.impulso.network/ruby-4-tipos-de-variaveis-que-voce-precisa-conhecer/
- Entrada e Saída Padrão
  - https://www.rubyguides.com/2018/10/puts-vs-print/
- String e Interpolação de Variáveis
  - https://ruby-doc.org/core-3.0.0/String.html
- Coerção | Casting
  - https://www.educative.io/answers/how-type-casting-or-type-conversion-is-done-in-ruby
- Criando Métodos
  - https://www.tutorialspoint.com/ruby/ruby_methods.htm#
- Range e Estruturas de Repetição
  - https://ruby-doc.org/core-2.5.1/Range.html
  - https://www.campuscode.com.br/conteudos/loops-em-ruby-com-metodos-times-e-upto
  - https://www.geeksforgeeks.org/ruby-loops-for-while-do-while-until/
- Hash
  - https://ruby-doc.org/core-3.1.0/Hash.html
- Array
  - https://ruby-doc.org/core-3.1.0/Array.html
  - https://apidock.com/ruby/Enumerable/reduce
  - https://learn.co/lessons/nested-arrays-ruby
  - https://medium.com/collabcode/diferen%C3%A7a-entre-map-collect-select-e-each-no-ruby-4d8dc853711f