



UNIVERSIDADE FEDERAL DO PIAUÍ  
CENTRO DE CIÊNCIAS DA NATUREZA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

---

1. Introdução ao Ruby

d. Constantes

- i. Não existe algo que seja realmente constante
- ii. Padrão: Variáveis declaradas com a primeira letra maiúscula
- iii. Não impede que o usuário altere seu valor
  1. O Ruby somente avisa que você não deveria estar fazendo isso
- iv. Ex
  1. `CONSTANTE = "asda"`
  2. `=> "asda"`
  3. `CONSTANTE = 1`
  4. `Constante = 2`
  5. `=> 2`
  6. `Constante = 5`

e. Intervalos numéricos

- i. Além de números é possível declarar intervalos numéricos
- ii. 2 tipos:
  1. Inclusivos: inclui o último número
  2. Exclusivos: não inclui o último número
- iii. Declarados usando `..` e `...`, respectivamente
- iv. Ex

1. `a = 1..10`
2. `=> 1..10`
3. `b = 1...10`
4. `=> 1...10`
5. `a.each do |v|`
6. `print "#{v} "`
7. `end`
8. `1 2 3 4 5 6 7 8 9 10 => 1..10`
9. `b.each do |v|`
10. `print "#{v} "`
11. `end`
12. `1 2 3 4 5 6 7 8 9 => 1...10`

f. Arrays

- i. Coleções de valores
- ii. Em Ruby, não são tipados: podem conter valores de diversos tipos
- iii. 2 formas de se declarar um array genérico
  1. `arr = []`
  2. `=> []`
  3. `arr = Array.new`
  4. `=> []`
- iv. Declaração especial de array de strings



UNIVERSIDADE FEDERAL DO PIAUÍ  
CENTRO DE CIÊNCIAS DA NATUREZA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

---

1. `arr = %w{a b c}`
  2. `=> ["a", "b", "c"]`
- v. Junto com hashes são as estruturas mais usadas do Ruby
1. Flexibilidade e facilidade de iteração
    - a. Métodos `each` e `each_with_index`
  2. Diversos outros métodos
    - a. `arr = %w{a b c}`
    - b. `=> ["a", "b", "c"]`
    - c. `arr.methods`
  3. Importantes
    - a. `select`: recebe um bloco e retorna um novo array com o elementos para o qual o bloco retornou `true`
    - b. `[]=`: define o valor de uma posição
    - c. `[]`: retorna o valor da posição passada
    - d. `last`: retorna o último item
    - e. `empty?`: verdadeiro se está vazio
    - f. `equal?`: comparação com outro array
    - g. `each_index`: recebe um bloco e passa apenas os índices do array para ele
    - h. `sort`: retorna um novo array com os itens ordenados
    - i. `sort!`: altera o array de origem
    - j. `+`: soma dois arrays, criando outro com itens de ambos
    - k. `-`: subtrai dois arrays, criando outro com os itens do primeiro que não estão no segundo
    - l. `push`: adiciona item no fim do array
    - m. `pop`: retorna o último item do array e o remove
    - n. `find`: recebe bloco com um parâmetro e retorna o primeiro elemento para o qual ele foi verdadeiro
    - o. `clear`: remove todos os itens do array
    - p. `shift`: retorna o primeiro item e o remove
    - q. `first`: retorna o primeiro item
    - r. `inject`: recebe valor inicial e um bloco com 2 parâmetros: valor atual e item atual do array. Retorna o resultado da operação realizada no bloco
  4. Ex
    - a. `arr = [1, 2, 3, 4, 5, 6]`
    - b. `=> [1, 2, 3, 4, 5, 6]`
    - c. `arr.inject(0) do |val, it|`
    - d. `val + it`
    - e. `end`
    - f. `=> 21`



UNIVERSIDADE FEDERAL DO PIAUÍ  
CENTRO DE CIÊNCIAS DA NATUREZA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

---

5. Existe outra forma de criar arrays: criar métodos com uma lista variável de parâmetros
  - a. Declarar o último parâmetro como \*nome
  - b. Ex
    - i. `def parametros_variaveis *arr`
    - ii. `if arr`
    - iii. `arr.each do |v|`
    - iv. `print "#{v.class} - #{v}\n"`
    - v. `end`
    - vi. `end`
    - vii. `end`
    - viii. `=> nil`
    - ix. `parametros_variaveis`
    - x. `=> []`
    - xi. `parametros_variaveis 1, "asda", :simb, :a =>`  
`"teste", :arr => %w{a b c}`
    - xii. `Fixnum – 1`
    - xiii. `String – asda`
    - xiv. `Symbol – simb`
    - xv. `Hash – atestarrabc`
    - xvi. `=> [1, "asda", :simb, {:a => "teste", :arr =>`  
`["a", "b", "c"]}]`
  - g. Hashes
    - i. Outra construção bastante utilizada
      1. Será bastante usada em rails
    - ii. Semelhantes a arrays, mas são coleções do tipo chave=valor
    - iii. São tão comuns que existe até um operador especial => (associado)
    - iv. 2 formas de se declarar
      1. `h = { 1 => "asda", "b" => 123 }`
      2. `=> { "b" => 123, 1 => "asda" }`
      3. `h1= {}`
      4. `=> {}`
      5. `h2 = Hash.new`
      6. `=> {}`
    - v. Também possui diversos métodos
      1. `h.methods.sort`
    - vi. Métodos importantes
      1. `[]`: Retorna o valor da chave passada como parâmetro
      2. `[]=`: Atribui o valor da chave
      3. `each`: Executa um bloco com 2 argumentos para cada posição



UNIVERSIDADE FEDERAL DO PIAUÍ  
CENTRO DE CIÊNCIAS DA NATUREZA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

---

4. `each_key`: Executa um bloco com 1 argumento (chave) para cada posição
  5. `each_value`: Executa um bloco com 1 argumento (valor) para cada posição
  6. `has_key?`: verdadeiro se a chave existe
  7. `has_value?`: verdadeiro se o valor existe
  8. `default=`: definir qual valor será retornado quando for buscada chave inconsistente
  9. `default_proc`: igual ao `default=` mas executa um bloco para criar o valor para novas chaves
  10. `delete`: remove o valor correspondente à chave
- h. Símbolos
- i. Nomes começados por :
  - ii. Muito usados como chaves em hashes
  - iii. São basicamente strings, mas ocupam muito menos processamento do interpretador e memória
    1. `to_sym` transforma strings em símbolos
  - iv. Usar símbolos é sempre melhor que strings (desempenho)
- i. Expressões Regulares
- i. Forma fácil de extrair informações de textos ou alterar textos com padrões razoavelmente complexos
  - ii. Fazem parte da linguagem Ruby (diferente de outros que onde é biblioteca externa)
  - iii. 3 formas de declarar
    1. `er = /(.*?)/`
    2. `er = %r{(.*)}`
    3. `er = Regexp.new "(.*)"`
  - iv. Instâncias da classe `Regexp`
  - v. Métodos
    1. `er = /^[0-9]/`
    2. `=> /^[0-9]/`
    3. `"123" =~ er`
    4. `=> 0`
    5. `er =~ "123"`
    6. `=> 0`
    7. `er =~ "abc"`
    8. `=> nil`
    9. `er !~ "123"`
    10. `=> false`
    11. `er !~ "abc"`
    12. `=> true`



UNIVERSIDADE FEDERAL DO PIAUÍ  
CENTRO DE CIÊNCIAS DA NATUREZA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

---

13. =~ procura pela expressão no texto e retorna a posição em que foi encontrada
  14. !~ Informa se existe uma ocorrência da expressão no texto
- j. Classes e métodos
- i. Classe: base da orientação a objeto
    1. Tudo em Ruby são objetos
  - ii. Classes também são objetos e podem ter métodos próprios
    1. Métodos de classe
    2. São herdados por objetos descendentes e podem saber a qual objeto pertencem
      - a. self aponta para a classe e não para uma instância
  - iii. Ex
    1. class Carro
    2. def initialize fabricante, modelo, ano
    3. @fabricante = fabricante
    4. @modelo = modelo
    5. @ano = ano
    6. end
    7. attr\_accessor :fabricante, :modelo, :ano
    8. end
    9. => nil
    10. clio = Carro.new "Renault", "clio", "2000"
    11. clio.modelo
    12. => "clio"
  - iv. Ruby suporta herança simples: operador <
    1. class Clio < Carro
    2. @@fabricante = "Renault"
    3. @@modelo = "clio"
    4. def initialize ano
    5. super @@fabricante, @@modelo, ano
    6. end
    7. end
    8. => nil
    9. clio = Clio.new 2003
  - v. Métodos de classe
    1. class Fabrica
    2. def self.clio
    3. Clio.new 2003
    4. end
    5. def self.megane
    6. Carro.new "Renault", "megane", 2003



UNIVERSIDADE FEDERAL DO PIAUÍ  
CENTRO DE CIÊNCIAS DA NATUREZA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

---

7. end
8. end
9. Fabrica.clio
10. Fabrica.megane
  - a. Notação self.metodo
  - b. Pode ser criada para classes ou para instâncias a qualquer momento
    - i. Classe anônima para aquele objeto
- vi. Métodos: Object.methods.sort
  1. class: classe do objeto
  2. class\_eval: executa string contendo código ruby no contexto da classe
  3. class\_variable\_defined?: Informa se variável está definida na classe
  4. const\_defined?: Informa se existe constante definida na classe
  5. const\_get: lê o valor de uma constante
  6. const\_set: grava o valor de uma constante ou cria uma nova
  7. instance\_eval: executa string contendo código ruby no contexto da instância de uma classe
  8. instance\_methods: lista todos os métodos de instância da classe
  9. instance\_variable\_defined? Informa se uma variável está definida para instâncias da classe
  10. instance\_variable\_get lê o valor de uma variável de instância
  11. instance\_variable\_set cria ou altera o valor de uma variável de instância
- k. Métodos
  - i. Não existem
- l. Módulos
- m. Operadores Condicionais e Loops
  - i. if / elsif / else / end
  - ii. case / when / else / end
- n. Operadores de Loop
  - i. while
  - ii. for
  - iii. until
  - iv. begin
  - v. loop
  - vi. Padrões importantes
- o. Dominando o Ruby