

# Linguagem Ruby



**PUC Minas**  
**Poços de Caldas**

Luiz Alberto Ferreira Gomes

Curso de Ciência da Computação

3 de setembro de 2020

# Ruby

---

## 1 Introdução

## 2 Entrada e Saída

## 3 Controle de Fluxo

## 4 Loops e Interações

# Ruby

---

- Linguagem inventada por Yukihiro "Matz" Matsumoto
- Versão 1.0 liberada em 1996(Japão)
- Popularizada no início de 2005 pelo Rails



Ruby

# Ruby

---

- Linguagem **dinâmica** e **orientada a objetos**
- Elegante, **expressiva** e declarativa
- Influenciada pelo Perl, Smalltalk, Eiffel e Lisp

## ..Java..

---

```
1  public class Print3Times {  
2      public static void main(String[] args) {  
3          for(int i = 0; i < 3; i++) {  
4              System.out.println("Hello World!")  
5          }  
6      }  
7  }
```

---

# ..Ruby..

---

## Listing 1: hello.rb

```
1 # um comentario em ruby
2 3.times { puts "Hello World" }
```

# Básico do Ruby

---

- Indentação de 2 espaços para cada nível aninhado (recomendado)
- # é utilizado para comentários
  - use com moderação, o código deve ser auto documentado
- Scripts utilizam a extensão .rb

## Listing 2: hello.rb

```
1  # um comentario em ruby
2  3.times { puts "Hello World" }
```

# Convenção de Nomes

---

## ■ Variáveis e Métodos

- em **minúsculas** e separada\_por\_sublinhado (tenha mais de uma palavra)
- métodos ainda permitem no final os caracteres ?!

## ■ Constantes

- tanto TODAS\_AS\_LETRAS\_EM\_MAIUSCULAS ou no formato CamelCase

## ■ Classes(e módulos)

- formato CamelCase



# Remoção do Ponto-e-Vírgula

---

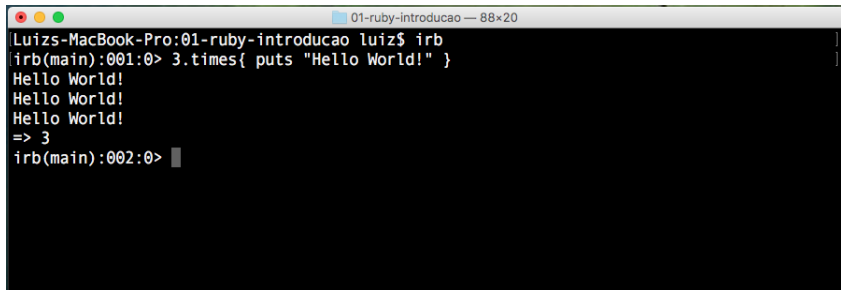
- Não coloque o ponto-e-vírgula no final da linha
- Pode ser utilizado para colocar várias declarações em uma linha
  - altamente desencorajado

```
1 a = 3
2 a = 3; b = 5
```

# Interactive Ruby (IRB) (1)

---

- Console **interativa** para interpretação de comandos Ruby
- Instalado com o interpretador Ruby
- Permite a **execução** de comandos rapidamente

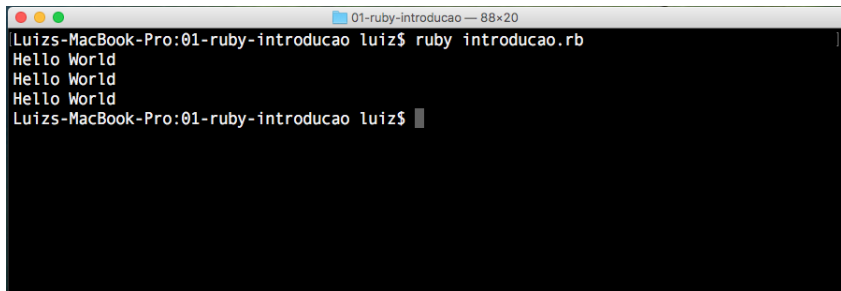


```
Luizs-MacBook-Pro:01-ruby-introducao luiz$ irb
irb(main):001:0> 3.times{ puts "Hello World!" }
Hello World!
Hello World!
Hello World!
=> 3
irb(main):002:0> 
```

## Interactive Ruby (IRB) (2)

---

- Permite a **execução** de **scripts** contendo vários comandos

A screenshot of a macOS terminal window titled "01-ruby-introducao — 88x20". The terminal shows a user at the prompt "Luiz-MacBook-Pro:01-ruby-introducao luiz\$" running the command "ruby introducao.rb". The script outputs "Hello World" three times. The prompt then returns to "Luiz-MacBook-Pro:01-ruby-introducao luiz\$".

```
Luiz-MacBook-Pro:01-ruby-introducao luiz$ ruby introducao.rb
Hello World
Hello World
Hello World
Luiz-MacBook-Pro:01-ruby-introducao luiz$
```

# Ruby

---

1 Introdução

**2** Entrada e Saída

3 Controle de Fluxo

4 Loops e Interações

# Entrada pelo Teclado

---

- `gets` é método **padrão para receber** um valor pelo teclado

```
1  # recebe um valor do tipo string.  
2  nome = gets
```

- Utilize `gets.chomp` para remover o caracter de nova linha.

```
1  # remove o caracter de nova linha.  
2  nome = gets.chomp
```

- Utilize `gets.chomp.to_i` para converter o valor lido para inteiro.

```
1  # converte a string recebida para inteiro.  
2  idade = gets.chomp.to_i
```

# Saída na Tela

---

- puts é método **padrão** para impressão em tela
  - insere uma quebra de linha após a impressão
  - similar ao `System.out.println` do Java

```
1  # exibe da tela do computador.  
2  puts "Informacoes do jogador"  
3  puts "Nome %s" % nome  
4  puts "Idade %d" % idade  
5  puts "Nome %s \nIdade %d" % [nome, idade]
```

## Hora de Colocar as Mão na Massa (1)

---

1. Escreva um *script* Ruby leia o nome de um jogador do jogo de adivinha é um palpite, ambos digitados pelo teclado. Após isto, o script deverá apresentar os valores lidos.

# Ruby

---

1 Introdução

2 Entrada e Saída

**3 Controle de Fluxo**

4 Loops e Interações



# Controle de Fluxo (1)

---

if ... elsif ... else  
unless  
case

## Controle de Fluxo (2)

---

- Não existe a necessidade de uso de parênteses ou chaves
- Utilize a instrução `end` no final do bloco

Listing 3: if.rb

```
1 nivel = 5
2 if nivel == 3
3   puts "basico"
4 elsif numero == 5
5   puts "avancado"
6 else
7   puts "desconhecido"
8 end
9 # => avancado
```

Listing 4: unless.rb

```
1 quantidade = 5
2 unless a == 6
3   puts "quantidade incorreta"
4 end
5 # => quantidade incorreta
```

# Controle de Fluxo (3)

---

## Listing 5: case\_1.rb

```
1 idade = 21
2 case
3   when idade >= 21
4     puts "Voce pode comprar cerveja"
5   when 1 == 0
6     puts "Escrito por um programador bebado"
7   else
8     puts "Nada a dizer"
9 end
10 # Vode pode comprar cerveja
```

# Controle de Fluxo (4)

---

Listing 6: case\_2.rb

```
1 nome = 'Otello Neves'
2 case nome
3   when /neve/i then puts "Algo e suspeito aqui"
4   when 'Eduardo' then puts "Seu nome e Eduardo"
5 end
6 # Algo e suspeito aqui
```

## Operadores Lógicos (em ordem de precedência)

---

---

<=, <, >, >=	Comparação
==, !=	Igual ou diferente
&&	Conectivo <b>e</b>
	Conectivo <b>ou</b>

---

# True e False

---

- **false** e **nil** são booleanos **FALSOS**
- Todo o restante é **VERDADEIRO**

Listing 7: true\_false.rb

```
1 puts "0 e true" if 0
2 puts "false e true?" if "false"
3 puts "nao - false e false" if false
4 puts "string vazia is true" if ""
5 puts "nil e true?" if "nil"
6 puts "nao - nil is false" if nil
```

# Hora de Colocar as Mão na Massa (1)

---

1. Escreva um *script* Ruby leia o nome de um jogador do jogo de adivinha é um palpite, ambos digitados pelo teclado. Após isto, o script deverá apresentar os valores lidos e se o usuário acertou o número secreto ou não.

# Recapitulando

---

- Existe muitas opções de fluxo de controle
- A formato em um linha é muito expressiva
- Exceto `nil` e `false`, os demais valores são verdadeiros.



# Ruby

---

1 Introdução

2 Entrada e Saída

3 Controle de Fluxo

**4** Loops e Interações

# Loops e Interações (1)

---

loop  
while e until  
for  
each e times

## Loops e Interações (2)

---

### ■ loop

Listing 8: loop.rb

```
1 i = 0
2 loop do
3   i += 2
4   puts i
5   if i == 10
6     break
7   end
8 end
9 # 2
10 # 4
11 # 6
12 # 8
13 # 10
```

# Loops e Interações (3)

---

## ■ while e until

Listing 9: while.rb

```
1 a = 10
2 while a > 9
3   puts a
4   a -= 1
5 end
6 # => 10
```

Listing 10: until.rb

```
1 a = 9
2 until a >= 10
3   puts a
4   a += 1
5 end
6 # => 9
```

## Loops e Interações (4)

---

- for (**difícilmente empregado**)
- each/times é preferível

Listing 11: for\_loop.rb

```
1 for i in 0..2
2   puts i
3 end
4 # => 0
5 # => 1
6 # => 2
```

# Loops e Interações (5)

---

## ■ each

Listing 12: each\_1.rb

```
1 nomes = ['Joao', 'Maria', 'Ana']
2 nomes.each { |nome| puts nome }
3 # Joao
4 # Maria
5 # Ana
```

Listing 13: each\_2.rb

```
1 nomes = ['Joao', 'Maria', 'Ana']
2 n = 1
3 nomes.each do |nome|
4   puts "#{n}.#{nome}"
5   n += 1
6 end
7 # 1.Joao
8 # 2.Maria
9 # 3.Ana
```

# Hora de Colocar as Mão na Massa (1)

---

1. Escreva um *script* Ruby leia o nome de um jogador do jogo de adivinha e apresente o valor lido.
2. O *script* Ruby deverá sortear um número de 1 a 10 e permite que o usuário tente 3 vezes até acertá-lo. A cada tentativa errada, o programa informa se o número a adivinhar está abaixo ou acima. **Dica:** utilize `rand(n) + 1`

# Solução do Exercício (1)

---

## Listing 14: loop.rb

```
1 puts "Bem-vindo ao jogo da adivinhacao"
2 puts "Qual e o seu nome?"
3 nome = gets
4 puts "\n\n\n\n"
5 puts "Comecaremos o jogo para voce, " + nome
6 puts "Escolhendo um numero secreto entre 1 e 10..."
7 numero_secreto = rand(10) + 1
8 puts "Escolhido... "
9 puts "Que tal adivinhar hoje o nosso numero secreto?"
10
11
12 (1..3).each do | tentativa |
13     puts "\n\n\n"
14     puts "Tentativa %d" %tentativa
15     puts "Entre com o numero"
```



## Solução do Exercício (2)

---

```
16 chute = gets.chomp.to_i
17 puts "áSer que acertou? êVoc chutou %d" % chute
18
19 acertou = numero_secreto == chute.to_i
20
21 if acertou
22     puts "Acertou!"
23     break
24 else
25     maior = numero_secreto > chute.to_i
26     if maior
27         puts "0 numero secreto e maior!"
28     else
29         puts "0 numero secreto e menor!"
30     end
31 end
32 end
```

# Recapitulando

---

- Existe muitas opções de loops e interações
- `each` é preferível ao loop `for` para percorrer arrays