Linguagem Ruby

PUC Minas
Poços de Caldas

Luiz Alberto Ferreira Gomes

Curso de Ciência da Computação

11 de setembro de 2020

Ruby

Funções e Métodos

- Tecnicamente, uma função é definida fora de uma classe
- Um método é definido dentro de uma classe
- Em Ruby, toda função/método é pertence a pelo menos uma classe
  - nem sempre explicitamente escrito em uma classe

Conclusão: Toda função é na verdade um método em Ruby

Métodos

- Parênteses são opcionais
  - tanto para definição quanto para a chamada do método
- Usado para tornar o código mais claro

parens.rb

```
1 def soma
2   puts "sem parenteses"
3 end
4 def subtrai()
5   puts "com parenteses"
6 end
7 soma()
8 soma
9 subtrai
```

Parâmetros e Retorno

- Não é necessário declarar o tipo dos parâmetros
- O método pode retornar qualquer valor
- O comando `return` é opcional
  - o valor da última linha executada é retornada

return_optional.rb

```ruby
def soma(um, dois)
  um + dois
end
def divide(um, dois)
  return "Acho que nao..." if dois == 0
  um / dois
end
puts soma(2, 2) # => 4
puts divide(2, 0) # => Acho que nao...
puts divide(12, 4) # => 3
```

Nomes de Métodos Expressivos

- Nomes de métodos podem terminar com:
  - '?' - métodos com retorno booleano
  - '!' - métodos com efeitos colaterais

Listing 1: expressive.rb

```ruby
def pode_dividir_por?(n)
  return false if n.zero?
  true
end
puts pode_dividir_por? 3 # => true
puts pode_dividir_por? 0 # => false
```

Argumentos Padrões(Defaults)

- Métodos podem ter argumentos padrões
  - se o valor é passado, ele é utilizado
  - senão, o valor padrão é utilizado

Listing 2: default_args.rb

```ruby
def fatorial(n = 5)
   n == 0? 1 : n * fatorial(n - 1)
end
puts fatorial 5  # => 120
puts fatorial    # => 120
puts fatorial(3) # => 6
```

Quantidade Variável de Argumentos

- * prefixa o parâmetro com quantidade variável de argumentos
- Pode ser utilizado com parâmetros no início, meio e final

Listing 3: splat.rb

```
1 def max(um_parametro, *numeros, outro)
2   # os argumentos do parametro numero se
3   # tornam um array
4   numeros.max
5 end
6 puts max("algo", 7, 32, -4, "algo mais") # => 32
```

Exercícios (1)

1. Refatore o jogo que permite o usuário adivinhar para utilizar métodos.

Listing 4: splat.rb

```
1  def da_boas_vindas
2    puts 'Bem-vindo ao jogo da adivinhacao'
3    puts 'Qual e o seu nome?'
4    nome = gets
5    puts "\n\n\n\n"
6    puts "Comecaremos o jogo para voce, #{nome}"
7  end
8
9  def sorteia_numero_secreto
10   puts 'Escolhendo um numero secreto entre 1 e 10...'
11   numero_secreto = rand(10) + 1
12   puts 'Escolhido... que tal adivinhar hoje o nosso numero secreto?
```

Exercícios (2)

```
13    numero_secreto
14  end
15
16  def pede_um_numero(tentativa, limite_de_tentativas)
17    puts "\n\n\n"
18    puts "Tentativa #{tentativa} de #{limite_de_tentativas}"
19    puts 'Entre com o numero'
20    chute = gets
21    puts "Sera que acertou? Voce chutou #{chute}"
22    chute.to_i
23  end
24
25  def verifica_se_acertou(numero_secreto, chute)
26    acertou = numero_secreto == chute
27    if acertou
28      puts 'Acertou!'
29      return true
30    else
```

Exercícios (3)

```
31      maior = numero_secreto > chute
32      if maior
33        puts 'O numero secreto e maior!'
34      else
35        puts 'O numero secreto e menor!'
36      end
37      false
38    end
39  end
40
41  def joga(limite_de_tentativas)
42    numero_secreto = sorteia_numero_secreto
43    (1..limite_de_tentativas).each do |tentativa|
44      chute = pede_um_numero tentativa, limite_de_tentativas
45      break if verifica_se_acertou numero_secreto, chute
46    end
47  end
48
```

Exercícios (4)

```
49 def nao_quer_jogar?
50   puts 'Deseja jogar novamente? (S/N)'
51   quero_jogar = gets.strip.upcase
52   quero_jogar.casecmp('N').zero?
53 end
54
55 da_boas_vindas
56 limite_de_tentativas = 3
57 loop do
58   joga limite_de_tentativas
59   break if nao_quer_jogar?
60 end
```

Recapitulando

- Não há necessidade de declarar o tipo de parâmetro passado ou retornado (linguagem dinâmica)
- `return` é opcional - a última linha executável é "retornada"
- Permite métodos com quantidade variável de argumentos ou argumentos padrão

Ruby

Blocos (1)

- Um "Trecho" de código
  - escrito entre chaves($\{\}$) ou entre do e end
  - passado para métodos como o último parâmetro
- **Convenção**
  - use chaves($\{\}$) quanto o bloco contém uma linha
  - use do e end quando o bloco contém múltiplas linhas
- Frequentemente utilizado em iteração

Blocos (2)

Listing 5: times.rb

```ruby
1.times { puts "Hello World!" }
# => Hello World!
2.times do |index|
  if index > 0
    puts index
  end
end
# => 1
2.times { |index| puts index if index > 0 }
# => 1
```

Utilizando Blocos

- Duas técnicas para utilizar blocos nos métodos
- **Implicitamente:**
  - use `block_given?` para checar se o bloco foi passado
  - use `yield` para chamar o bloco
- **Explicitamente:**
  - use `&` como prefixo do último parâmetro
  - use `call` para chamar o bloco

Técnica Implícita (1)

- Necessário checar com `block_given?`
  - se não uma excessão será lançada

Listing 6: implicit_blocks.rb

```
1  def totaliza(valores)
2    return "Nenhum bloco foi passado" unless block_given?
3    total = 0
4    for valor in valores
5      total += valor
6      yield(total)
7    end
8  end
9
10 totaliza([ 20, 30, 40, 10 ]){| resultado | puts resultado }
11 totaliza([ 20, 30, 40, 10 ]) do | resultado |
12   resultado = resultado * 0.25
```

Técnica Implícita (2)

```
13    puts "#{resultado}"
14  end
15
16  puts totaliza ([ 20, 30, 40, 10 ]) # => Nenhum bloco foi passado
```

Técnica Explícita (1)

- Necessário checar com `nil?`

Listing 7: implicit_blocks.rb

```
1  def totaliza(valores, &um_bloco)
2    return "Nenhum bloco foi passado" if um_bloco.nil?
3    total = 0
4    for valor in valores
5      total += valor
6      um_bloco.call(total)
7    end
8  end
9
10 totaliza([ 20, 30, 40, 10 ]){| resultado | puts resultado }
11 totaliza([ 20, 30, 40, 10 ]) do | resultado |
12   resultado = resultado * 0.25
13   puts "#{resultado}"
14 end
```

Técnica Explícita (2)

```
15
16 puts totaliza ([ 20, 30, 40, 10 ]) # => Nenhum bloco foi passado
```

Recapitulando

- Blocos são apenas trechos de códigos que podem ser passados para métodos
- Tanto explicitamente quanto implicitamente

Ruby

Strings (1)

- Strings com aspas simples
  - permitem a utilização de ' com \
  - mostra a string como foi escrita
- Strings com aspas duplas
  - interpreta caracteres especiais como \n e \t
  - permite a interpolação de strings, evitando concatenação

Strings (2)

Listing 8: strings.rb

```
1  aspas_simples = 'D\' Silva Filho\n programa em Ruby!'
2  aspas_duplas  = "D\' Silva Filho\n programa em Ruby!"
3  puts aspas_simples # => D' Silva Filho\n programa em Ruby!
4  puts aspas_duplas  # => D' Silva Filho\n
5                     # =>  programa em Ruby!
6  def multiplica (um, dois)
7    "#{um} multiplicado por #{dois} = #{um * dois}"
8  end
9  puts mutiplica(5, 3)
10 # => 5 multiplicado por 3 = 15
```

Strings (3)

- Métodos terminados com ! modificam a string
  - a maioria retorna apenas um novo string
- Permite o uso do %Q{textos longos com multiplas linhas}
  - o mesmo comportamento de strings com aspas duplas
- É essencial dominar a API de Strings do Ruby

Strings (4)

Listing 9: more_strings.rb

```
1  nome = " tim"
2  puts nome.lstrip.capitalize # => Tim
3  p nome # => " tim"
4  nome.lstrip! # remove os espacos do inicial (modifica)
5  nome[0] = 'K' # substitui o primeiro caracter
6  puts nome # => Kim
7
8  clima = %Q{O dia esta quente la fora
9                pegue os guarda\-chuva}
10
11 clima.lines do |line|
12   line.sub! 'quente', 'chuvoso' # substitui 'quente' with 'chuvoso'
13   puts "#{line.strip}"
14 end
15 # =>  dia esta quente la fora
16 # =>       pegue os guarda\-chuvas
```

Símbolos

- :simbolo – string altamente otimizadas
  - ex. :domingo, :dolar, :calcio, :id
- Constantes que não precisam ser pré-declaradas
- Garantia de unicidade e imutabilidade
- Podem ser convertidos para uma String com to_s
  - ou de String para Símbolo com to_sym

Recapitulando

- A interpolação evita a concatenação de strings
- Strings oferecem uma API muito útil

Exercícios (1)

1. Refatore o jogo de adivinhar nos seguintes métodos:
   - **da_boas_vindas** que dá boas vindas e retorna o nome do usuário.
   - **sorteia_numero_screto** que retorna o número secreto sorteado.
   - **pede_um_numero** retorna o numero digitado pelo usuário.
   - **verifica_se_acertou** retorna um booleano se o usuário acertou ou não o número secreto.
   - **nao_quer_jogar** retorna se o usuário apos as tentativas quer jogar outra vez ou não.
   - **joga** método que joga as três tentativas chamando os métodos necessários.