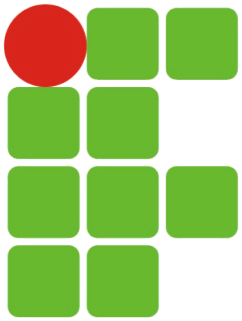

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE
Campus Natal - Central

Introdução ao Ruby

Fellipe Aleixo (*fellipe.aleixo@ifrn.edu.br*)

Ruby

- Linguagem orientada a objetos
  - Tudo o que você manipula em Ruby é um objeto
  - O resultado dessas manipulações também são objetos

- Modelagem orientada a objetos → modelar conceitos do “mundo real”
  - Classes → das quais são gerados os objetos

Orientação a Objetos

- Um objeto é a combinação de
  - Estado (atributos e seus valores)
  - Métodos (operam sobre o estado)

- São criados por meio de um construtor
  - Construtor padrão = **new()**

```
line_item_one = LineItem.new
line_item_one.quantity = 1
line_item_one.sku = "AUTO_B_00"
```

Métodos

- Métodos são invocados pelo envio de uma mensagem para um objeto
  - Mensagem = nome do método [+ parâmetros]

```
"dave".length
line_item_one.quantity()
cart.add_line_item(next_purchase)
submit_tag "Add to Cart"
```

  - Os parênteses são, geralmente, opcionais na chamada dos métodos

Nomenclatura

- (i) variáveis locais, (ii) parâmetros de método e (iii) nomes de métodos devem iniciar com uma letra minúscula ou um “underline” (_)
  - Exemplos: order, line_item e xr2000
- Atributos (variáveis de instância) devem iniciar com um arroba (@)
  - Exemplos: @quantity e @product_id
  - “underline” é usado para unir várias palavras

Nomenclatura

- (i) classes, (ii) módulos e (iii) constantes devem iniciar com uma letra maiúscula
  - União de várias palavras com letras maiúsculas
  - Exemplos: Object, PurchaseOrder e LineItem
- Rails utiliza símbolos para identificar coisas
  - Nomeando parâmetros de métodos

```
redirect_to :action => "edit", :id => params[:id]
```

Métodos

- Definição de funções que, opcionalmente, (i) recebem parâmetros e (ii) geram resultados

```
def say_goodnight(name)
   result = 'Good night, ' + name
   return result
end

# Time for bed...
puts say_goodnight('Mary-Ellen')          # => 'Goodnight, Mary-Ellen'
puts say_goodnight('John-Boy')            # => 'Goodnight, John-Boy'
```

  - Caracter “#” defini um comentário

Tipos de Dados

- ***Strings***
  - Pode ser criado a partir de *strings* literais
    - Separados por aspas simples (‘) ou duplas (“)
    - Para os *strings* entre aspas duplas – Ruby trabalha um pouco mais
      - Primeiro procura por substituições (Ex.: \n)
      - Depois faz a interpretação de expressões (#{expressão})

```
def say_goodnight(name)
   "Good night, #{name.capitalize}"
end
puts say_goodnight('pa')
```

Arrays e *Hashes*

- Possibilitam trabalhar com coleções de objetos indexados (acessados por um chave)
  - Nos *arrays* a chave é um número inteiro
  - Nos *hashes* a chave é um objeto qualquer
- Crescem para armazenar novos elementos

```
a = [ 1, 'cat', 3.14 ]       # array with three elements
a[0]                         # access the first element (1)
a[2] = nil                   # set the third element
                             # array now [ 1, 'cat', nil ]
```

Arrays e *Hashes*

- O método “<<()” é utilizado com *arrays*

```
ages = []
for person in @people
   ages << person.age
end
```

- Atalhos para a criação de *arrays*

```
a = [ 'ant', 'bee', 'cat', 'dog', 'elk' ]
# this is the same:
a = %w{ ant bee cat dog elk }
```

Arrays e *Hashes*

- Nos *hashes* são utilizadas chaves para a inserção e recuperação de elementos

```
inst_section = {
    :cello => 'string',
    :clarinet => 'woodwind',
    :drum => 'percussion',
    :oboe => 'woodwind',
    :trumpet => 'brass',
    :violin => 'string'
}
```

  - "=>" é opcional

Expressões Regulares

- Uma expressão regular permite a especificação padrão de caracteres
  - Para localizar a mesma em uma *string*
  - Criado com **/pattern/** ou **%r{pattern}**

```
if line =~ /P(erl|ython)/
   puts "There seems to be another scripting language here"
end
```

Estruturas de Controle de Fluxo

- Instrução **if**

```
if count > 10
   puts "Try again"
elsif tries == 3
   puts "You lose"
else
   puts "Enter a number"
end
```

Estruturas de Controle de Fluxo

- Instrução **while**

```
while weight < 100 and num_pallets <= 30
   pallet = next_pallet()
   weight += pallet.weight
   num_pallets += 1
end
```

- Algumas variantes precisam de atenção
  - Ex.: **unless** e **until**
  - Expressões em uma única linha

Estruturas de Controle de Fluxo

- Expressões em uma linha

```
puts "Danger, Will Robinson" if radiation > 3000

distance = distance * 1.2 while distance < 100
```

- Algumas variantes precisam de atenção
  - Ex.: **unless** e **until**
  - Expressões em uma única linha

Blocos e Iteradores

- Blocos de código

```
{ puts "Hello" }              # utilizado geralmente em blocos de uma linha

do                            ###
   club.enroll(person)            # também representa um bloco
   person.socialize               #
end                           ###
```

  - Pode ser passado um bloco para um método

```
greet { puts "Hi" }
# passando também parâmetros
verbose_greet("Dave", "loyal customer") { puts "Hi" }
```

Instrução **yield**

- Chamada de um bloco associado
  - Podem ser passados parâmetros para o bloco – nominados entre barras verticais (|)
- Blocos são comumente usados com iteradores

```
animals = %w( ant bee cat dog elk )          # cria um array
animals.each {|animal| puts animal }         # itera no seu conteúdo

3.times { print "Ho! " }                      #=> Ho! Ho! Ho!
```

Instrução **yield**

- Capturando e nomeando um bloco passado para um método

```
def wrap &b
   print "Santa says: "
   3.times(&b)
   print "\n"
end

wrap { print "Ho! " }
```

Exceções

- Objeto da classe **Exception**, ou uma subclasse
- Interrompe o fluxo normal de execução

```
begin
   content = load_blog_data(file_name)
rescue BlogDataNotFound
   STDERR.puts "File #{file_name} not found"
rescue BlogDataFormatError
   STDERR.puts "Invalid blog data in #{file_name}"
rescue Exception => exc
   STDERR.puts "General error loading #{file_name}: #{exc.message}"
end
```

Classes

- Estrutura similar a outras linguagens OO
  - Exemplo:

```
class Order < ActiveRecord::Base
   has_many :line_items
   def self.find_all_unpaid
      self.where('paid = 0')
   end
   def total
      sum = 0
      line_items.each {|li| sum += li.total}
      sum
   end
end
```

Classes

- As variáveis de instância são precedidas de @

```
class Greeter
   def initialize(name)
      @name = name
   end
   def name
      @name
   end
   def name=(new_name)
      @name = new_name
   end
end

g = Greeter.new("Barney")
```

Classes

- Métodos acessadores e modificadores

```
class Greeter
   attr_accessor :name        # create reader and writer methods
   attr_reader :greeting      # create reader only
   attr_writer :age           # create writer only
end
```

Classes

- Visibilidade de métodos

```
class MyClass
   def m1
   end
   protected
   def m2
   end
   private
   def m3
   end
end
```

Módulos

- Similares a classes, visto que definem uma (i) coleção de métodos, (ii) constantes, (iii) definição de classes ou (iv)outros módulos

- Dois propósitos principais
  - Atuam como “namespaces”
  - Compartilhar funcionalidades entre classes

YAML

- Acrônimo de *YAML Ain't Markup Language*
- Utilizada para a definição de configurações

```
development:
    adapter: sqlite3
    database: db/development.sqlite3
    pool: 5
    timeout: 5000
```

Convertendo Objetos

- Ruby permite converter um objeto em um *stream* de bytes
  - Visando o armazenamento do mesmo
  - Processo denominado de *marshaling*

- Rails utiliza esse processo para armazenar os dados das sessões dos usuários

DICA:

VALE A PENA BUSCAR MAIS INFORMAÇÕES DA LINGUAGEM RUBY