Orientação a Objetos com Ruby

Arthur de Moura Del Esposte - esposte@ime.usp.br

Apresentação

- Engenheiro de Software pela **Universidade de Brasília**
- Mestrando em Ciência da Computação pelo **IME - USP**
- Pesquisa sobre Arquiteturas Distribuídas em Plataformas de Cidades Inteligentes
- Desenvolvedor de **Software Livre**
- Outros interesses incluem: **Música, Colecionismo, Viagens,** ect...

Código de Qualidade

Maior Produtividade

Expressividade e Agilidade

Divertir enquanto Programa

Código de Qualidade

Maior Produtividade

Expressividade e Agilidade

Divertir enquanto Programa

Livre

Matura

Grande Comunidade

Extensível

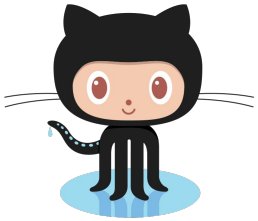
O que o seguinte código faz?

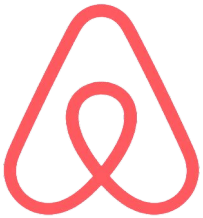
```
5.times { print "Hello!" }
```

Quem usa Ruby?

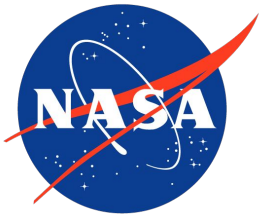
DevOps

Web





Simulações



Mais histórias de sucesso em Ruby

Bibliografia

- Armando Fox, David Patterson. **Construindo Software como Serviço: Uma Abordagem Ágil Usando Computação em Nuvem.** **(Capítulo 3)**
- Peter Cooper. **Beginning Ruby: From Novice to Professional**
- Sandi Metz. **Practical Object-Oriented Design in Ruby (POODR).**
- Jim Gay. **Clean Ruby.**
- Russ Olsen. **Design Patterns in Ruby.**

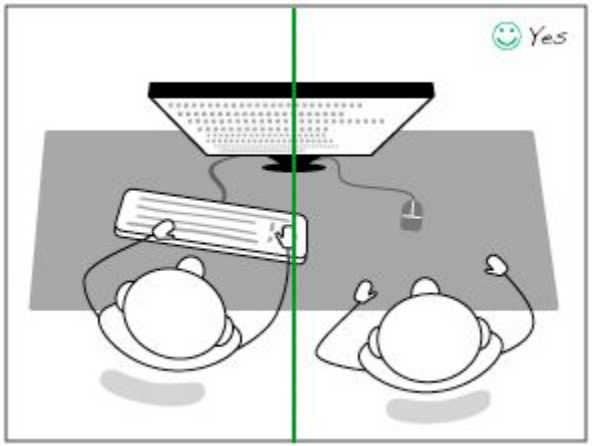
Recursos Online

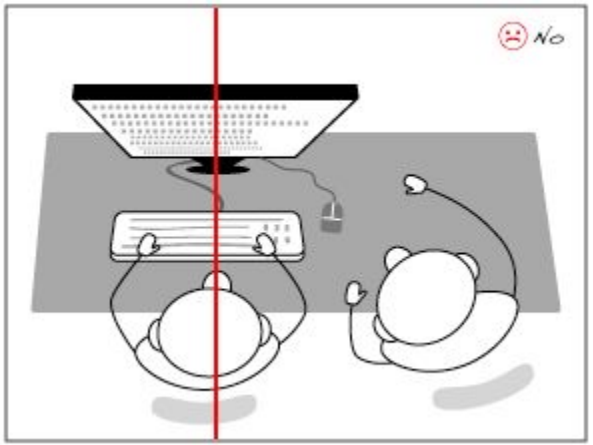
- Documentação: http://ruby-doc.org/
- Guia de Programação: http://ruby-doc.com/docs/ProgrammingRuby/
- Ruby GEMs: https://rubygems.org/
- Useful tools: https://github.com/markets/awesome-ruby
- Cursos Online:
  - https://www.codesdope.com
  - http://tryruby.org/
  - https://www.edx.org/course/agile-development-using-ruby-rails-uc-berkeleyx-cs169-1x
  - https://www.codecademy.com/pt-BR/learn/ruby
  - https://learnrubythehardway.org/book/

Programação em Pares

- Produtividade
- Evita distrações
- Colaboração
- Comunicação constante
- Nivelamento no aprendizado

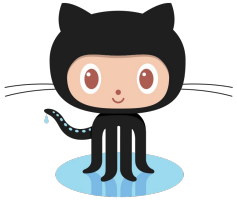
Programação em Pares





Contato

https://gitlab.com/arthurmde



https://github.com/arthurmde

http://bit.ly/2jvND12

http://bit.ly/2j0llo9

Centro de Competência em Software Livre - CCSL

esposte@ime.usp.br

Obrigado!

Aula 01 - Introdução a Ruby

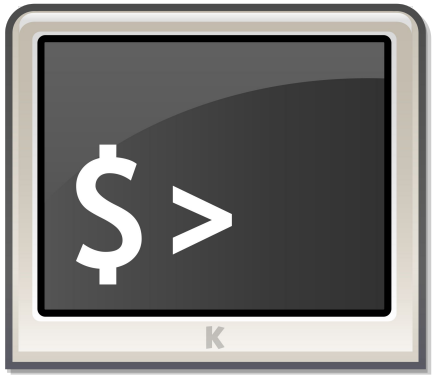
Arthur de Moura Del Esposte - esposte@ime.usp.br

Agenda

- Ambiente de desenvolvimento
- Introdução a Ruby
- Conceitos básicos de Ruby
- Estilo, convenções e Tipos básicos
- Array e Hash

Ambiente de Desenvolvimento

O que precisamos?



Terminal

Ruby Environment

Editor de Texto

Instalação do Ruby

- Linux:
  - $ sudo apt install ruby-full
  - $ sudo yum install ruby
- Para Windows: http://rubyinstaller.org/
- **RVM: https://rvm.io/rvm/install**

Ruby Version Manager

- Instalar e trabalhar com múltiplas versões
- Gerenciar conjunto de bibliotecas (GEMs)
- Instalação do RVM:

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3

$ \curl -sSL https://get.rvm.io | bash -s stable

$ /bin/bash --login
```

Ruby Version Manager

- Listar versões de Ruby disponíveis:

```
$ rvm list
```

- Instalar uma versão específica do Ruby:

```
$ rvm install 2.3.0
```

- Utilizar a versão instalada:

```
$ rvm use 2.3.0
```

- Verificar instalação e versão do Ruby:

```
$ ruby -v
```

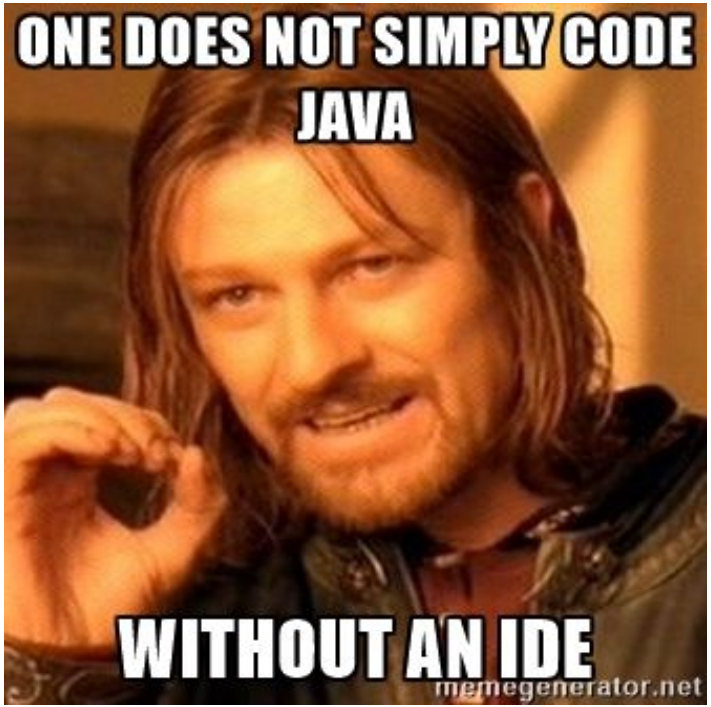
- Verificar local de instalação do Ruby:

```
$ which ruby
```

- Utilizar a versão instalada:

```
$ rvm use 2.3.0
```

Qual IDE devo usar?



Integrated Development Environment

- Editor de Textos
- Automatização de **Compilação**
- Debugging
- **Auto-complete?**

Qual IDE devo usar?

Editor de Textos

IDE

- Ruby Mine
- Cloud9

Editores de Texto

- VIM
- EMACS
- Gedit
- **Atom**
- Sublime

Mais sobre escolhas de IDE vs Editor de Textos

A linguagem Ruby

Um pouco de história...

“Eu queria uma linguagem interpretada que fosse mais poderosa do que Perl e mais orientada a objetos do que Python.”

- Disponibilizada em 1995 por Matz
- Popularizada com a criação do Rails em 2005
- Versão 2.4 (Natal de 2016)

Yukihiro "Matz" Matsumoto

Ruby

- Interpretada
- De Scripts até Sistemas de Produção
- Agrega conceitos de diversas linguagens
  - Perl
  - Python
  - Smalltalk
  - Lisp
- Multiparadigma

Ruby

- Orientada a Objetos
  - Tudo é um **objeto**
  - Toda operação é uma chamada de **método** de algum **objeto**
- Dinamicamente Tipada
- Dinâmica e Extensível
  - Adicionar e modificar código em tempo de execução (**Metaprogramação**)
  - Perguntar aos objetos sobre eles (**Reflexão**)

Tudo é um objeto!

```
5.times { print "Hello!" }
```

Interactive Ruby Shell - IRB

Abrir o IRB!

$ /bin/bash --login

$ rvm use default

$ irb

Testes no IRB

```
> 1

> 1.class

> 1.0.class

> 3 + 6

> 1.methods

> 1.to_f

> -2.positive?

> String.class

> String.superclass
```

Scripts Ruby

Scripts Ruby

- Usamos scripts Ruby (arquivos **.rb**) em vez de usar diretamente o interpretador interativo do Ruby (**IRB**).
- Portanto, crie uma nova pasta com um arquivo .rb dentro:

```
$ mkdir exercise_01

$ cd exercise_01

$ touch hello.rb

$ atom hello.rb
```

Scripts Ruby

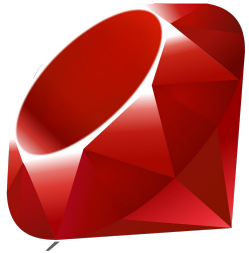
- Escreva no arquivo:

```
1  #!/usr/bin/env ruby
2
3  puts "Hello World!"
```

- Execute o arquivo:

```
$ ruby hello.rb
```

Scripts Ruby



- Para dividir o código em unidades lógicas menores, é necessário incluir arquivos outros arquivos usando:
  - **load**: inclui o arquivo **toda vez** que o método é executado
  - **require**: inclui o arquivo **apenas uma vez**

```
load 'filename.rb'

require 'filename'
```

- Para incluir os nossos arquivos, vamos usar quase sempre **require_relative**

Primeiros Passos

Covenções - Parte 1

- Código sempre em **Inglês**
- Classes usam **UpperCamelCase**
- Metodos & Variáveis usam **snake_case**
- **Não existe declaração de variáveis!**
  - Variáveis locais devem ser atribuídas antes de usadas
  - Correto: x = 3; x = ‘foo’
  - Errado: Integer x = 3

Manipulação básica de objetos

- Criação de instâncias de uma classe:
  - `favorite_song = Song.new("A Sort of Homecoming")`
- Chamada de métodos
  - `radio.play(favorite_song) # => Radio#play`
  - `Radio.find("Kiss FM") # => Radio.play`

Expressividade e Métodos

- Os parênteses são opcionais nas chamadas de método
  - `radio.play favorite_song`
  - `person.add_parents father, mother`
- Métodos que retornam **true** ou **false** terminam com uma **interrogação** em seu nome
  - `radio.has? favorite_song`
- Métodos que alteram o objeto que foi chamado terminam com uma **exclamação** (Método Perigoso)
  - `foo = "DOWNCASE"`
  - `foo.downcase # => returns a new string`
  - `foo.downcase! # => modifies the foo object`

Testes no IRB

Convenções e símbolos

```
> “hello”.class

> “hello”.methods

> TestConstant = 1

> TestConstant = 2

> soccer_team = :cruzeiro

> :cruzeiro = "something"

> :cruzeiro.methods

> :cruzeiro == "cruzeiro"# => false
```

Covenções - Parte 2

- **Comentários** usam **#**
- **Constantes** (escopo) & **$globais** (sem escopo)
- **Símbolos** *:valid_symbol*
  - Strings imutáveis cujos valores são elas mesmas
  - Seguem convenções de variáveis
  - Normalmente se usa quando você precisa de uma String que não será impressa na tela

Comparadores Básicos

- **true**
- **false**
- **nil**
- Operadores:
  - ==
  - !=
  - < >
  - <= & >=
  - && ||

Strings & I/O

Strings

- Cadeias de caracteres dentro de **aspas** simples ou dupla
  - “Conhecendo ruby...”
  - ‘2112’
  - “Pode-se usar ‘aspas simples’ aqui”
  - ‘Enquanto podemos usar “aspas duplas” aqui’

Os métodos **puts** & **print** podem ser utilizados para imprimir strings no console

Strings

- Variáveis podem ser incorporadas em strings com aspas duplas:

```ruby
name = 'Arthur'
age = 24

puts "My name is #{name} and I am #{age} years old"
# => "My name is Arthur and I am 24 years old"

puts 'My name is #{name} and I am #{age} years old'
# => "My name is #{name} and I am #{24} years old"
```

Testes no IRB

Operações com Strings

```
> "Ho! " * 3

> "Ho! " * 0

> country = "brazil"

> "Hello from " + country

> country.capitalize!

> "".empty?

> "I love Rock".include?("I love")
```

Testes no IRB

Operações com Strings 2

```
> a = "world"

> a.prepend("hello ")

> "stressed".reverse

> x = String.new("X")

> x.methods
```

Entrada de Dados

- Enquanto o **puts** imprime uma linha na saída padrão, usamos o **gets** para ler a próxima linha da entrada padrão.
- O **gets** retorna a *String* lida

```
1  line = gets
2  print line
3
4  line_size = gets.size
5  print line_size
```

Entrada de Dados

- **chomp** é um método de String que pode ser utilizado para remover o **\n** do final

```
1  print "How old are you? "
2  age = gets.chomp
3  print "How tall are you? "
4  height = gets.chomp
5  print "How much do you weigh? "
6  weight = gets.chomp
```

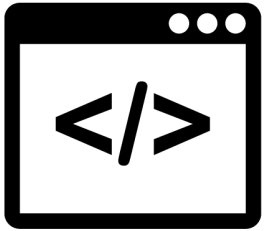
Quais expressões atribuem um inteiro (Fixnum) na variável ‘number’?

1. `number = gets`
2. `number = gets.to_i`
3. `number = gets.chomp`
4. `number = gets.chomp.to_i`

Quais expressões atribuem um inteiro (Fixnum) na variável ‘number’?

1. `number = gets`
2. ✓ `number = gets.to_i`
3. `number = gets.chomp`
4. ✓ `number = gets.chomp.to_i`

Exercício



Escreva um script Ruby que receba o nome completo de um usuário e imprima a seguinte frase:

*“O nome **Arthur de Moura Del Esposte** tem **27** caracteres e um total de **2** ocorrências da letra ‘a’”*

- Obs:
  - *“Arthur de Moura Del Esposte”* foi o nome recebido como entrada do usuário
  - Veja os métodos disponíveis de String para tentar usar algum que ajude!

Array & Hash

Coleções

- **Arrays** e **Hashes** são coleções indexadas de objetos que são acessados através de uma **chave**
  - **Array:** chaves são inteiros
  - **Hash:** chaves são quaisquer objetos
- Os objetos da coleção podem ser de diferentes tipos

Array - Acesso de elementos

```ruby
number = [1, 2, 3, 4, 5, 6]
number[2]     #=> 3
number[100]   #=> nil
number[-3]    #=> 4
number[2, 3] #=> [3, 4, 5]
number[1..4] #=> [2, 3, 4, 5]

number.first #=> 1
number.last  #=> 6
number.first(3) #=> [1, 2, 3]
number.last(3)  #=> [4, 5, 6]
```

```
number = [1, 2, 3, 4, 5, 6]
```

Qual a saída de **`number[1..-3]`** ?

1. `[1, 2, 3, 4]`
2. `[1, 2, 3]`
3. `[2, 3, 4]`
4. `[]`

```
number = [1, 2, 3, 4, 5, 6]
```

Qual a saída de **`number[1..-3]`** ?

1. `[1, 2, 3, 4]`
2. `[1, 2, 3]`
3. ✓ `[2, 3, 4]`
4. `[]`

Array - Manipulação elementos

```ruby
[1,2,3] << "a" # => [1,2,3,"a"]
[1,2,3].push("a") # => [1,2,3,"a"]
[1,2,3].unshift("a") # => ["a",1,2,3]

my_array = [1,2,3]
my_array.pop # => 3
my_array # => [1,2]
my_array = ["a",1,2,3]
my_array.shift # => "a"
my_array # => [1,2,3]
my_array.insert(1, 'apple')  #=> [1, 'apple', 2, 3]

my_array = [:a, :b, :c, :b]
my_array.delete(:b) # => :b
my_array # => [:a, :c]
```

```
number = [1, 2, 3, 4, 5, 6]
```

Qual a saída de **`number << [4,5,6]`** ?

1. `[1, 2, 3, 4, 5, 6, [4, 5, 6]]`
2. `[1, 2, 3, 4, 5, 6, 4, 5, 6]`
3. `[1, 2, 3]`
4. `[1, 2, 3, 4, 5, 6]`

```
number = [1, 2, 3, 4, 5, 6]
```

Qual a saída de **`number << [4,5,6]`** ?

1. `[1, 2, 3, 4, 5, 6, [4, 5, 6]]`
2. `[1, 2, 3, 4, 5, 6, 4, 5, 6]`
3. `[1, 2, 3]`
4. `[1, 2, 3, 4, 5, 6]`

Descreve a saída para as operações com o Array:
my_array = [:a, [1, 2, 3], *true*, *true*, *false*]

```
1.  my_array.size
2.  my_array.include? 1
3.  my_array.include? "a"
4.  my_array.include? false
5.  my_array.count true
6.  my_array[3]
7.  my_array[1, 2]
8.  my_array[1][2]
```

Descreve a saída para as operações com o Array:
my_array = [:a, [1, 2, 3], *true*, *true*, *false*]

```
1.  my_array.size # => 5
2.  my_array.include? 1 # => false
3.  my_array.include? "a" # => false
4.  my_array.include? false # => true
5.  my_array.count true # => 2
6.  my_array[3] # => true
7.  my_array[1, 2] # => [[1,2,3], true]
8.  my_array[1][2] # => 3
```

Iteração

- Nunca queira escrever um Loop '**for**' tradicional para acessar os elementos de uma coleção em Ruby!
- Um '**for**' não é orientado a objetos!
- Cada coleção define métodos para iterar sobre seus elementos:

```ruby
["first", "middle", "last"].each { |element| puts element.capitalize }

["first", "middle", "last"].reverse_each { |element| puts element.upcase }

["a", "b", "c"].each_with_index do |letter, index|
  puts "#{letter.upcase}: #{index}"
end
```

Seleção

- Você pode selecionar elementos de um Array de acordo com uma condição

```ruby
my_array = [1, 2, 3, 4, 5, 6]
my_array.select { |a| a > 3 }      #=> [4, 5, 6]
my_array.reject { |a| a < 3 }      #=> [3, 4, 5, 6]
my_array.drop_while { |a| a < 4 } #=> [4, 5, 6]
my_array                           #=> [1, 2, 3, 4, 5, 6]
```

Hash - Parte 1

- Dicionários ou Hashmaps =)
- Armazena uma coleção de elementos como uma lista, mas são acessados por **chaves** de vários tipos, não apenas por índices numéricos como o Array.
- **Hashes** associam um objeto **A** com outro objeto **B**, onde a chave **A** mapeia para o valor **B:**

  {A => B}

Documentação detalhada sobre Hash em Ruby

Hash - Parte 2

- Uma Hash é representada por **{ }**
- Exemplos:
  - `a = {}`
  - `a = Hash.new`
  - `a = {name: 'Ze', age: 39}`
  - `a = {'name' => 'Ze', 'age' => 39, 'height' => 6 * 12 + 2}`
  - `a = [3.14, "math", true]`

Testes no IRB

Manipulando Hashes

```
> ages = {}

> ages["José"] = 56

> ages["Maria"] = 32

> ages

> ages["Matheus"]

> person = {name: "José", age: 56}

> person[:name]

> person.keys

> person.methods
```

Descreve a saída para as operações com a Hash:

```
my_hash = {number: 1, colors: ["blue","red"], name: "Maria"}
```

1. `my_hash.size`
2. `my_hash.has_key? 1`
3. `my_hash.has_key? "number"`
4. `my_hash.has_key? :name`
5. `My_hash.has_value? "Maria"`
6. `my_hash[:number]`
7. `my_hash[:colors][1]`

Descreve a saída para as operações com a Hash:

```
my_hash = {number: 1, colors: ["blue","red"], name: "Maria"}
```

1. `my_hash.size # => 3`
2. `my_hash.has_key? 1 # => false`
3. `my_hash.has_key? "number" # => false`
4. `my_hash.has_key? :name # => true`
5. `My_hash.has_value? "Maria" # => true`
6. `my_hash[:number] # => 1`
7. `my_hash[:colors][1] # => "red"`

Iteração em Hash

- Hashes também oferecem métodos para iterar sobre seus elementos, semelhante aos Arrays

```ruby
my_hash = { "a" => 100, "b" => 200 }

my_hash.each {|key, value| puts "#{key} is #{value}" }
my_hash.each_key {|key| puts key }
my_hash.each_value {|value| puts value }

```

Revisão!

O que já vimos!

- Ambiente de desenvolvimento
- Introdução a Ruby
- Conceitos básicos de Ruby
- Estilo, convenções e Tipos básicos
- Array e Hash

Atividades Sugeridas!

Resolver os seguintes desafios

1. Dada uma frase inserida pelo usuário, informar quantas vezes cada letra do alfabeto apareceu:
   - **Entrada**: “Meu nome é Arthur”
   - **Saída**: a = 1, b = 0, c = 0, d = 0, e = 3, f = 0, ...
2. Dada uma frase inserida pelo usuário, informar quais palavras ocorreram no máximo uma vez
   - **Entrada**: “Todo mundo morre, mas nem todo mundo vive”
   - **Saída**: morre, mas, nem, vive

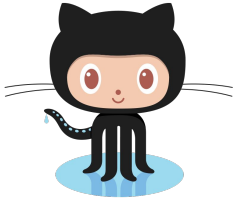
Estudar

- Explorar estruturas de dados no IRB, manipulando os métodos de Strings, Arrays, Hashes, e outros tipos básicos
- Explorar opções de entrada de dados e saídas (puts, print, gets)

Obrigado!

Contato

https://gitlab.com/arthurmde



https://github.com/arthurmde

http://bit.ly/2jvND12

http://bit.ly/2j0llo9

Centro de Competência em Software Livre - CCSL

esposte@ime.usp.br