



# Ruby

## Conceitos básicos

Prof. Ms. Leonardo Botega

Diego Gabriel Pereira



# Quem somos nós

- Leonardo Botega
  - Bacharel em Ciência da Computação – UNIVEM
  - Mestre em Ciência da Computação – UFSCar
  - Doutorando em Ciência da Computação – UFSCar
  - Docente dos cursos de Ciência da Computação e Sistemas de Informação do UNIVEM
- Diego Gabriel Pereira
  - Discente de Sistemas de Informação - UNIVEM

# Tópicos

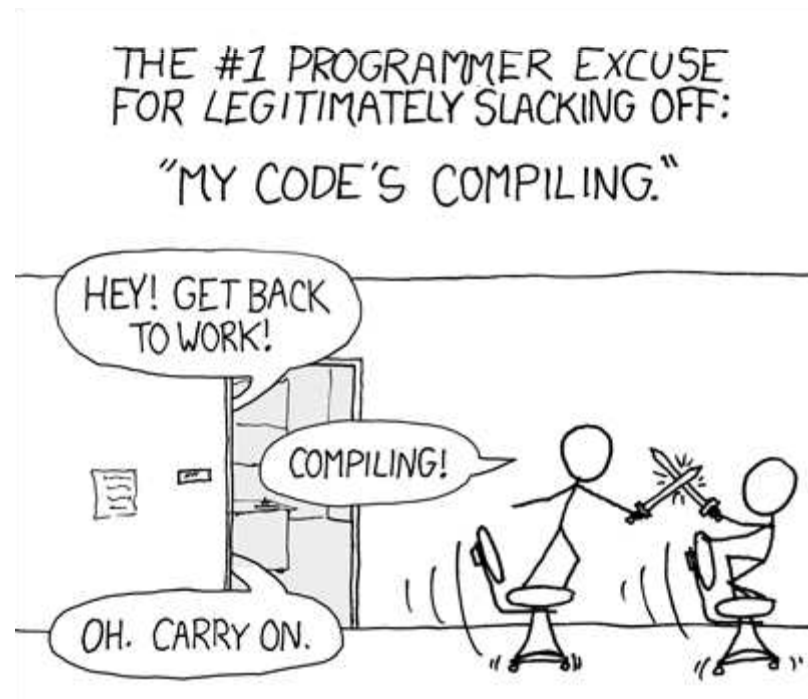
- O que é Ruby?
- História
- Características
- Programando em Ruby
- Implementações do Interpretador Ruby
- Onde posso usar Ruby?
- Plataformas suportadas
- Mercado de trabalho
- Referências

# O que é Ruby?

- Uma linguagem de programação:
  - Interpretada;
  - Open-Source;
  - Multiparadigmática;
  - Dinâmica.

# O que é Ruby?

- Interpretada:
  - Não é necessário compilar o código...



# O que é Ruby?

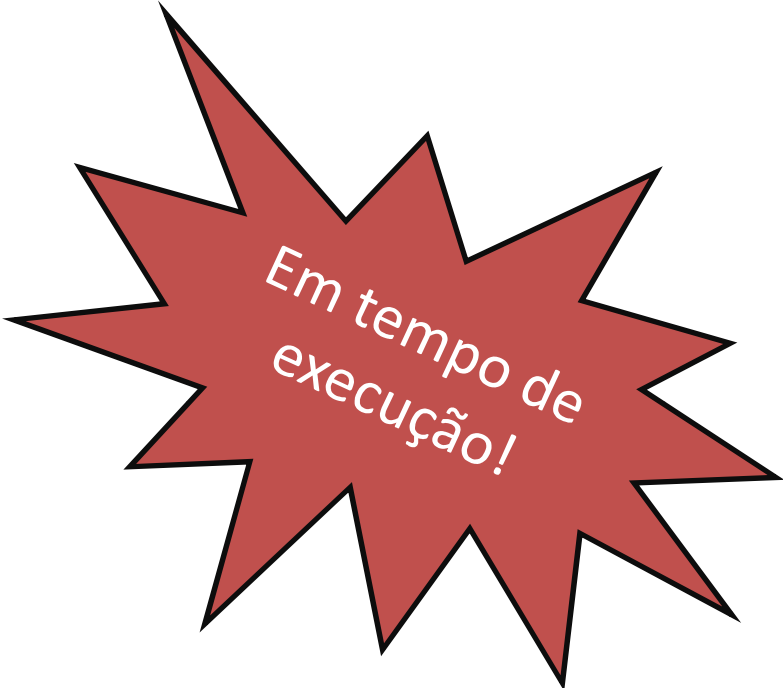
- Open-Source:
  - Você pode:
    - Olhar o código;
    - Estudar o código;
    - Alterar o código;
    - Melhorar o código!

# O que é Ruby?

- Multiparadigmática:
  - Orientada a objetos;
  - Funcional;
  - Permite programação imperativa.

# O que é Ruby?

- Dinâmica:
  - Adicionar novo código;
  - Extender objetos;
  - Extender classes;
  - Alterar sistema de tipos.



Em tempo de  
execução!



# História

- Criada no Japão por Yukihiro Matsumoto (Matz);
- Fevereiro/1993 - idéias iniciais
- Versão 0.95 lançada no Japão em dezembro/1995.
- Versão 1.0 lançada oficialmente em
- dezembro/1996.
- Versão atual (stable) - Ruby 1.8.7



# História

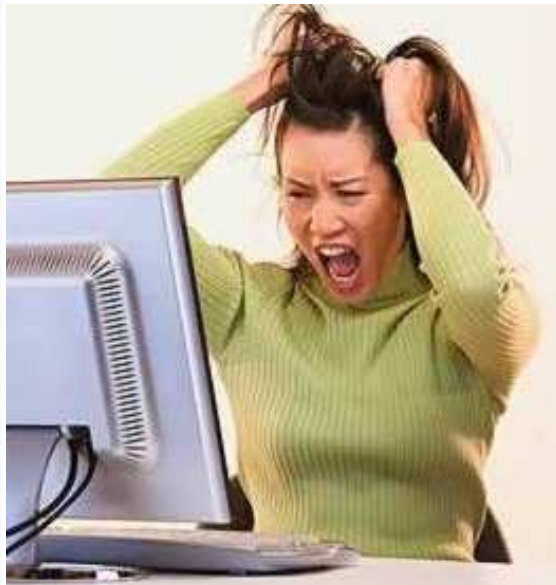
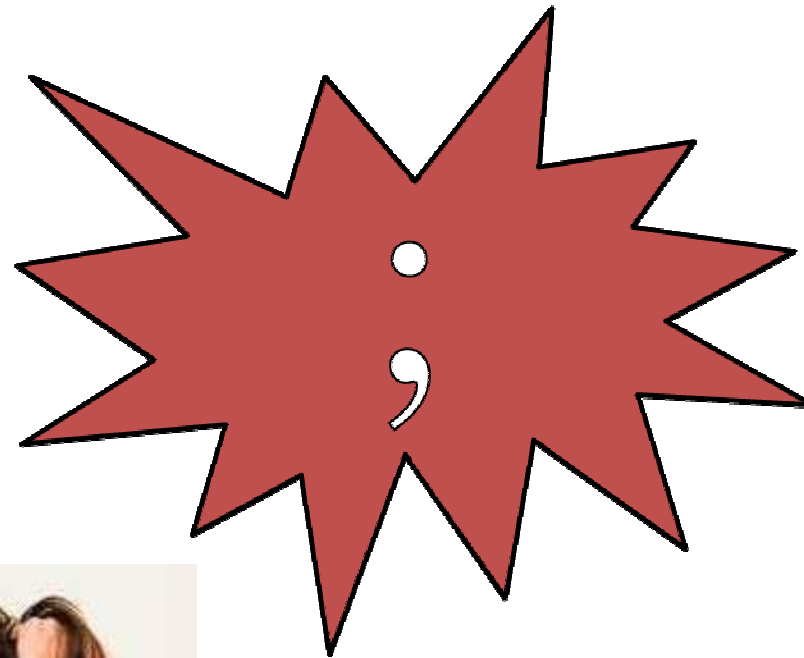
- Mas... Mais uma linguagem!?!? Pra quê?
  - Matz queria uma linguagem de script...
    - Mais poderosa que Perl;
    - Mais orientada a objetos que Python;
    - Que tivesse tudo o que ele sempre amou em Lisp, Eiffel e Smalltalk;
    - Funcional como Lisp, Haskell e Scheme.

# Desafio!

- Qual erro de compilação do código abaixo?

```
#include <stdio.h>
int main()
{
    printf("Erro? aonde?")
    return 0;
}
```

# Solução



# Solução!

- O Ruby não tem ponto e vírgula!



# Características

- Expressividade
  - Diga muito;
  - Seja claro;
  - Escreva menos código.

# Características

- Imprimir somente pares entre 1 e um número qualquer:
  - Linguagem C:

```
#include <stdio.h>
void imprime_pares(int limite)
{
    int i;
    for(i = 1; i <= limite; i++)
        if(i%2 == 0)
            printf("%d\n", i);
}
```

# Características

— Java:

```
public void imprime_pares(int limite) {  
    for(int i = 1; i <= limite; i++)  
        if(i%2 == 0)  
            System.out.println(i);  
}
```



# Características

— Ruby:

```
def imprime_pares(limite)
  1.upto(limite) {|i| puts i if i%2 == 0}
end
```

# Características

- Em Ruby, tudo é um objeto:

```
3.times { puts "Olá!" }  
# Olá!  
# Olá!  
# Olá!
```

```
puts "Ruby eh supimpa!".split.reverse  
# ==> ["supimpa!", "eh", "Ruby"]
```

```
true.class # <= TrueClass  
false.class # <= FalseClass  
nil.class # <= NilClass  
0.123.class # <= Float
```

# Características

- Os métodos SEMPRE retornam algo, nem que seja nil. Em geral, o valor retornado é o resultado da última expressão do método.

```
def um_metodo
end
```

```
um_metodo
# nil
```

```
def um_metodo
  1 == 2 - 1
end
```

```
um_metodo
# true
```

```
def um_metodo
  "Opa!"
end
```

```
um_metodo
# "Opa!"
```

```
def um_metodo
  (1..10).to_a
end
```

```
um_metodo
#
[1,2,3,4,5,6,7,8,9,10]
```

# Características

- Retornos múltiplos:

```
def um_metodo
  return 1, 2, 3
end

a, b, c = um_metodo

puts a, b, c
# 1
# 2
# 3
```

# Características

- Classes

```
class Logger
  def initialize(file)
    @messages_file = file
  end

  def add_message(msg)
    @messages_file << msg + "\n"
  end
end
```

# Características

```
puts "com um arquivo..."
log_file = File.new('logfile.txt', 'r+')
logger = Logger.new(log_file)
logger.add_message("uma mensagem")
logger.add_message("outra mensagem")
log_file.rewind
puts log_file.readlines.join
```

```
# uma mensagem
# outra mensagem
```

```
puts "com uma string"
log_string = String.new
logger = Logger.new(log_string)
logger.add_message("uma mensagem")
logger.add_message("outra mensagem")
puts log_string
```

```
# uma mensagem
# outra mensagem
```

# Características

- Metaprogramação

```
class Foo
end

Foo.class_eval do
  define_method("blabla") do |arg|
    puts arg
  end
end

f = Foo.new
f.blabla("123")
# 123
```

# Características

```
class Blergh
  def call_private_method
    private_method
  end

  private
  def private_method
    puts "Sou um método privado..."
  end
end
```

```
b = Blergh.new
b.call_private_method
# Sou um método privado...
```

```
b.private_method
# meta.rb:15: private method
# `private_method'
# called for #<Blergh:0xb7c3e940>
(NoMethodError)
```

```
b.send(:private_method)
# Sou um método privado...
```



# Características

```
class Foo
  def method_missing(name)
    puts "Nao sei responder a #{name.to_s} :-(  
  end
end
```

```
f = Foo.new
f.blabla
# Nao sei responder a blabla :-(
```

# Características

- Tempo de Compilação x Tempo de Execução
  - Em Ruby é tudo mesma coisa!

# Programando em Ruby

- Instalando:
  - Windows: One Click Installer
    - [ruby-lang.org](http://ruby-lang.org)
    - O instalador possui:
      - Interpretador;
      - IDE (SciTE)
      - GEM

# Programando em Ruby

- Primeiro programa:

```
puts "Hello World! =)"
```

# Programando em Ruby

- Cálculos:

```
puts 8 + 2 #adição
```

```
=> 10
```

```
puts 5 - 2 #subtração
```

```
=> 3
```

```
puts 5 * 3 #multiplicação
```

```
=> 15
```

```
puts 5 / 3 #divisão de inteiro
```

```
=> 1
```

```
puts 7.0/3 #divisão de fração
```

```
=> 2.3333333333333333
```

# Programando em Ruby

- Variáveis:

```
x = 7
```

```
puts x
```

```
=> 7
```

```
puts x.class
```

```
=> Fixnum
```

```
texto = "um texto qualquer"
```

```
puts texto
```

```
=> um texto qualquer
```

```
puts texto.class
```

```
=> String"
```

# Programando em Ruby

- Leitura e Escrita:

```
puts "Informe o seu nome:"
```

```
=> Informe o seu nome
```

```
nome = gets
```

```
puts "O seu nome é: " + nome
```

```
=> O seu nome é Diego
```

# Programando em Ruby

- Arrays:

```
a = [10, 20, 30, 55] #array de inteiro
```

```
puts a
```

```
b = ["Ruby", "é", "d+"] #array de string
```

```
puts b
```

```
c = %w(eu amo ruby) #array de string
```

```
puts c
```



# Programando em Ruby

- Ordenação de arrays:

```
a = [1, 3, 5, 4, 2]
```

```
puts a
```

```
=> 1, 3, 5, 4, 2
```

```
puts a.sort
```

```
=> 1, 2, 3, 4, 5
```

```
b = a.sort
```

```
puts b.reverse
```

```
=> 5, 4, 3, 2, 1
```

# Programando em Ruby

- Métodos:

```
def ola  
  puts "Hello World"  
end
```

```
ola  
=> Hello World
```

# Programando em Ruby

- **Classes:**

```
class Aluno
```

```
  attr_accessor :nome, :idade
```

```
end
```

```
a = Aluno.new
```

```
a.nome = "Michel"
```

```
a.idade = 20
```

```
puts "Nome: " + a.nome
```

```
=> Nome: Michel
```

```
puts "Idade: " + a.idade
```

```
=> TypeError: can't convert Fixnum into String
```

```
puts "idade: " + a.idade.to_s
```

```
=> Idade: 18
```

# Programando em Ruby

- Métodos de classe:

```
class Fabrica
```

```
  def self.clio  
    Clio.new(2003)  
  end
```

```
  def self.megane  
    Carro.new "Renault", "megane", 2003  
  end  
end
```

```
Fabrica.clio  
Fabrica.megane
```

# Programando em Ruby

- Variáveis de Instância e variáveis de classe:
  - Instância: definidas por @
  - Classe: definidas por @@

```
class Carro
  def initialize(fabricante, modelo, ano)
    @fabricante = fabricante
    @modelo = modelo
    @ano = ano
  end
  attr_accessor :fabricante, :modelo, :ano
end
```

```
clio=Carro.new "Renault", "clio", "2000"
clio.modelo
```

# Programando em Ruby

```
class Carro
  def initialize(fabricante, modelo, ano)
    @fabricante = fabricante
    @modelo = modelo
    @ano = ano
  end
  attr_accessor :fabricante, :modelo, :ano
end
```

```
class Clio < Carro
  @@fabricante = "Renault"
  @@modelo = "clio"
  def initialize (ano)
    super(@@fabricante, @@modelo, ano)
  end
end
```

```
clio= Clio.new(2003)
```

# Programando em Ruby

- Módulos: são similares a classes, mas não podem ser instanciados, e não podem herdar nem serem herdados, mas podem ser incluídos em classes. Ao ser incluído, instâncias da classe que o inclui são também do "tipo" do módulo incluído;
- Simulam a herança múltipla.

# Programando em Ruby

- Módulos:

```
module M
  def faz_algo
  end
end
class C
  include M
end
a = C.new
puts a.is_a?(M) # >> true
```



# Programando em Ruby

- Estruturas de Controle: Parecidas com a maioria das linguagens

```
if .. [then]          # if tradicional
[elsif .. [then] ..]
[else ..]
end
```

```
unless .. [then]      # negação do if (a menos que ...)
[else ..]
end
```

# Programando em Ruby

- Estruturas de Controle:

case .. # execução por casos

[when .. [,...] [,...]

[then] ..]

[else ..]

end

while .. [do] # while tradicional (faça enquanto ...) ..

end

# Programando em Ruby

- Estruturas de Controle:

`until .. [do] # negação do while (até que ...) .. end`

`for i in .. [do] # iteração por uma lista de objetos ..  
end`

# Programando em Ruby

- Estruturas de controle: if, unless, while e until podem ser usadas como modificadores, da seguinte forma:

```
puts "oi!" if permissao==true
```

```
puts i.to_s until i==0
```

# Programando em Ruby

- Iteradores: Iteradores são métodos que executam um bloco de código que lhes seja passado. Blocos são delimitados por { ... }, ou do ... end ({} tem maior precedência), e podem receber argumentos, declarados entre | ... |.

```
“diego”.each_byte { |c| print c, “ “}
```

# Implementações do interpretador Ruby

- MRI - Matz Ruby Interpreter
- Jruby - Implementação em Java do interpretador Ruby
  - Permite utilizar bibliotecas Java dentro do código Ruby
- Rubinius - Implementação em C++ e Ruby do interpretador Ruby
- IronRuby - Implementação em .NET do interpretador Ruby

# Onde posso usar o Ruby?

- Desktop;
- Web;
- Computação Científica;
- Bioinformática;
- Inteligência Artificial;
- Scripting;
- Testes Automatizados;
- Entre outros...

# Plataformas Suportadas

- Mac OS X (all varieties)
- Linux (all varieties)
- MS-DOS
- BSDs (including FreeBSD and OpenBSD)
- Acorn RISC OS
- Microsoft Windows 95, 98, XP, and Vista (all varieties)
- OS/2
- Amiga
- Celulares Symbian Series 60
- Qualquer plataforma para a qual exista uma Máquina Virtual
- Java (Usando JRuby ao invés do interpretador Ruby oficial)



# Mercado de Trabalho

- Ainda não é tão grande quanto o mercado de trabalho para Java, .NET, etc
- O maior uso do Ruby no Brasil e no mundo é através do framework Web Ruby on Rails.
- No Brasil, ainda cresce mais lentamente do que nos EUA e na Europa. O Brasil demora a aceitar novas tecnologias.
- Tem sido uma das linguagens mais utilizadas por start-ups de tecnologia

# Referências

- Ruby on Rails (Rodrigo Urubatan) – Editora Novatec;
- Linguagem de Programação Ruby (David Flanagan) – Editora Alta Books/O´Reilly;
- Ruby on Rails: Executando (Bruce A. Tate) – Editora Alta Books/O´Reilly.

# Exercício

- Crie uma classe Funcionários com os atributos “registro” e “função”;
- Dentro da classe Funcionários, crie pelo menos dois métodos;
- Crie pelo menos dois módulos (Ex: Pessoa e Colaborador) para comprovar a herança múltipla da classe Funcionários.