



1. Introdução ao Ruby

m. Métodos

i. Não existem

1. Não se chama método: envia-se mensagem a um objeto, que pode ter parâmetros e sempre tem um retorno
2. Métodos inexistentes podem ser adicionados no momento em que se tornem necessários
 - a. `method_missing`

ii. Ex

1. `class Teste`
2. `def method_missing method, *args`
3. `print "Método #{method} chamado na classe Teste, com argumentos #{args.join(', ')}\n"`
4. `end`
5. `end`
6. `t = Teste.new`
7. `t.imprimir`
8. `t.qualquer_coisa 1, 2, 3, "asd", :teste => 1`

iii. Não é uma boa prática usar o `method_missing` o tempo todo

1. `t.respond_to? :imprimir`
 - a. `false`, porém responde à mensagem (inconsistente)

iv. Cria-se a definição com `define_method` no momento em que um método inexistente se torna necessário

1. `module Propriedades`
`def propriedade nome`
`ivarname = "@#{nome}".to_sym`
`self.send :define_method, nome do`
`self.instance_variable_get ivarname`
`end`
`self.send :define_method, "#{nome}=" .to_sym do |val|`
`self.instance_variable_set ivarname, val`
`end`
`end`
`end`

n. Módulos

i. Repositórios de coisas

1. Podem conter classes
2. Podem ser usados como pacotes, para organizar classes em domínios muito grandes
3. Podem conter métodos para serem usados como "mixins"
 - a. Junto com classes abertas dão toda a flexibilidade que o Ruby possui



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

ii. Organizando classes

```
1. module Administracao
  class Cliente
    attr_accessor :nome
    def initialize
      @nome = ""
    end
  end
end
```

iii. Usando como mixins: declara-se apenas métodos no módulo

```
1. class Teste
  def ola_mundo
    print "Olá mundo\n"
  end
  def self.ola_mundo
    print "Olá mundo da classe\n"
  end
end
Teste.ola_mundo
t = Teste.new
t.ola_mundo
module MixinTest
  def self.included base
    base.send :include, InstanceMethods
    base.send :extend, ClassMethods
  end
  module ClassMethods
    def metodo_de_classe
      print "Novo método de classe definido no módulo
        'ClassMethods'\n"
    end
  end
  module InstanceMethods
    def metodo_de_instancia
      print "Novo método de instancia definido no
        módulo 'InstanceMethods'\n"
    end
  end
end
class Teste
  include MixinTest
end
Teste.metodo_de_classe
t.metodo_de_instancia
```

iv. Em qualquer classe podemos chamar include passando um módulo como parâmetro e os métodos estarão disponíveis



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

1. extend: métodos disponíveis para a instância
 2. send: envia mensagens para objetos (no caso, a classe Test)
- o. Operadores Condicionais e Loops
- i. Ruby também é imperativa (controle de fluxo e loop)
 - ii. if / elsif / else / end
 1. a = 0
if a == 0
 print "zero"
elsif a == 1
 print "um"
else
 print "não sei que número é esse"
end
zero => nil
b = 5 if a != 1
=> 5
 2. unless else end
 - a. unless = if not (facilita a leitura de código)
 - b. a = 1
=> 1
unless a == 0
 print "não é zero\n"
else
 print "é zero\n"
end
não é zero
=> nil
b = 6 unless b
=> 6
b = 7 unless b
=> nil
 - iii. case / when / else / end
 1. Atalho para sequência de elsif
 2. Mais flexível que Java ou C++
 - a. Pode ser usado com qualquer tipo de objeto
 - b. Só não é possível misturar
 3. Ex
 - a. a = 5
=> 5
case a
when 1..3



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

```
        puts "a entre 1 e 3\n"
when 4
    puts "a = 4\n"
else
    puts "nenhuma das anteriores\n"
end
nenhuma das anteriores
=> nil
a = "b"
case a
when "a"
    puts "a\n"
when "b"
    puts "b\n"
else
    puts "outra letra\n"
end
b
=> nil
```

p. Operadores de Loop

i. Podem ser usados com qualquer das estruturas de loop

1. break: sai do loop atual
2. next: executa o próximo passo
3. return: sai do loop e do método atual
4. redo: reinicia o loop atual

ii. while

1.

```
i = %w{a b c d e f}
=> ["a", "b", "c", "d", "e", "f"]
while b = i.pop
puts b
end
f
e
d
c
b
a
=> nil
```

2. Apenas um exemplo, nesse caso é mais indicado usar o método each da classe Array

iii. For



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

```
1. for i in 1..5
  puts i
end
1
2
3
4
5
=> 1..5
for a in %w{a b c d}
  puts a
end
a
b
c
d
=> ["a", "b", "c", "d"]
```

iv. until

```
1. Contrário do while
2. i = 4
   => 4
   until i == 0
     puts i
     i -= 1
   end
4
3
2
1
=> nil
```

v. begin

```
1. Em conjunto com while ou until, para fazer o teste depois
   a. Execução do código pelo menos uma vez
2. i = 0
   => 0
   begin
     puts i
     i += 1
   end while i < 0
0
=> nil
```



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

vi. loop

1. Executa até encontrar um break ou return
 - a. Não há condições
2. loop do
puts "a"
break if true
end
a
=> nil

vii. Padrões importantes

1. Esqueçam as linguagens anteriores
2. Nomes de arquivos
 - a. Letras minúsculas e _ para separar palavras
 - b. classe ClienteEspecial = cliente_especial.rb
 - c. Se estiver definida no modulo Clientes deverá estar dentro da pasta clientes
 - d. Não é imposição, apenas sugestão
3. Classes, atributos e métodos de acesso
 - a. Classe com letra maiúscula: também define-se uma constante
 - b. Métodos de acesso: attr_accessor
 - i. class Teste
attr_accessor :nome
end
t = Teste.new
t.methods.sort
4. Nomenclatura de métodos
 - a. Todas as letras minúsculas e palavras separadas por _
 - b. Métodos que transformam objetos começam por to_
 - i. to_s, to_i, to_a, to_sym

q. Dominando o Ruby

- i. Com o que foi apresentado seremos bons programadores rails, mas não bons programadores Ruby
 1. Bastante recomendado ler livro sobre Ruby
- ii. Todos os exemplos podem ser copiados em um arquivo .rb e executados com o comando **ruby**

2. Ambiente de desenvolvimento

a. Multiplataforma

i. Aptana RadRails

1. Baseada no eclipse para AJAX, DHTML e JavaScript
2. Subprojeto RadRails



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

3. Indicado para quem está acostumado a trabalhar no eclipse
 4. Tive alguns problemas de configuração no linux
 5. Snippets pobre
- ii. NetBeans
 1. Suporte a ruby on rails a partir da versão 6.1
 - a. Existe uma versão para quem quer apenas RoR, sem precisar de todo o suporte a Java EE
 2. Atalhos de código no estilo TextMate aumentam a velocidade de desenvolvimento
 3. É a que utilizo atualmente
 - iii. IntelliJ IDEA
 1. IDE Java paga
 - a. Única que sobreviveu à dupla Eclipse\Netbeans
 2. Na versão 7 adicionado suporte a RoR
 3. Dizem que é boa...
 - iv. Vim
 1. Editor de texto clássico do mundo Unix
 2. Plugins podem fazer até o vim enviar e-mail
 - a. Snippets e Code completion
 - b. rails.vim
 3. Extremamente leve
- b. Windows
 - i. Notepad++
 1. Bastante leve
 2. Poucos plugins
 3. Integração com o windows explorer
 4. Bastante usado como segundo editor
 - ii. E-TextEditor
 1. Clone do textmate para windows
 2. Integração com o cygwin
 3. Bundles e Snippets do textmate
 - c. Linux
 - i. Gedit
 1. Um dos mais rápidos para linux
 - a. Padrão do gnome
 2. Bastante extensível
 - a. É possível deixá-lo bastante parecido ao textmate
 - ii. Kate
 1. Bastante leve e personalizável
 - a. Disponível nas distribuições com KDE
 2. Tem muitos recursos disponível do Kdevelop



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

- d. Mac
 - i. Textmate
 1. É o preferido pela maioria dos desenvolvedores RoR
 2. Personalizável via bundles
 3. Tornou famosos os Code Snippets
 - a. Para diversas linguagens