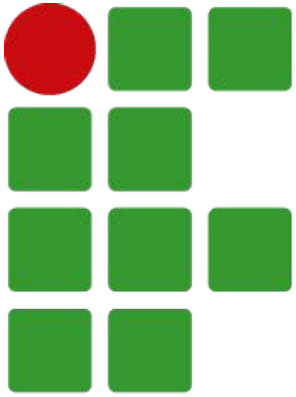


INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Linguagem Ruby

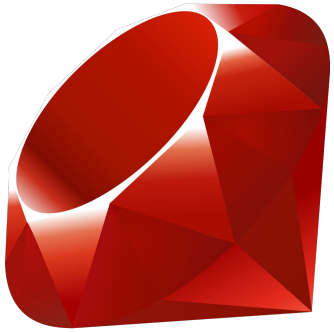
Desenvolvimento Web III

Prof. Diego Stiehl

INSTITUTO FEDERAL

Ruby

- Site oficial: https://www.ruby-lang.org
- Linguagem de Programação
  - Interpretada
  - Multiparadigma
    - Funcional, orientada a objetos, imperativa e reflexiva
  - Tipagem dinâmica e forte
  - Multiplataforma
- Criada para ser uma linguagem de script



INSTITUTO FEDERAL

Ruby - Histórico

- 1995 - Criação (documentada em japonês)
  - Por Yukihiro "Matz" Matsumoto
- 2000 - Tradução para inglês
  - Por Dave Thomas e Andy Hunt
  - Expansão da linguagem
- 2003 - Criação do framework Ruby on Rails
  - Por David Heinemeier Hansson (DHH)
  - Surgiu da criação do app 37signals
  - Proporcionou o “boom” da linguagem Ruby

INSTITUTO FEDERAL

Ruby - Características

- Open Source
  - Muitas bibliotecas disponíveis
- Foco na simplicidade e produtividade
- Código de leitura natural
  - Mais humano, menos máquina
- Código auto documentado

INSTITUTO FEDERAL

Exemplos

```ruby
# O Ruby sabe o que você
# quer dizer, mesmo que isso
# seja fazer contas em
# um Array completo
cidades    = %w[ Londres
                 Oslo
                 Paris
                 Amsterdã
                 Berlim ]
visitadas = %w[Berlim Oslo]

puts "Ainda me falta " +
     "visitar " +
     "as cidades:",
     cidades - visitadas
```

```ruby
# Output "Eu gosto de ruby"
diz = "Eu gosto de Ruby"
puts diz

# Saída "EU *GOSTO* DE RUBY"
diz['gosto'] = "*gosto*"
puts diz.upcase

# Output: "Eu *gosto*
# de Ruby" 5 vezes
5.times { puts diz }
```

INSTITUTO FEDERAL

IRB

- Interactive Ruby Shell
- Permite execução direta de comandos Ruby
  - Console
- Executar:
  - irb
- Para sair
  - CTRL + D ou digitar “quit”

INSTITUTO FEDERAL

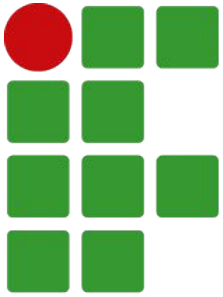
Pry

- http://pryrepl.org
- É uma alternativa ao IRB (padrão do Ruby)
- Oferece alguns recursos que facilitam o uso
  - Syntax highlighting
  - Indentação
  - Aceita plugins
  - Busca por documentação
  - ...
- `gem install pry`

INSTITUTO FEDERAL

Arquivo .rb

- Extensão padrão de arquivos contendo código-fonte escrito em Ruby
  - .rb
    - app.rb | teste.rb | main.rb
- Para execução em terminal
  - ruby [arquivo]
    - ruby sistema.rb



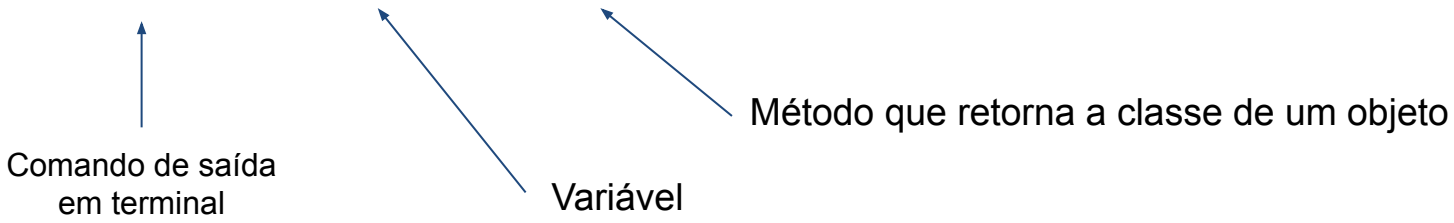
INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Descobrindo características da Linguagem Ruby

A Linguagem

Inferência de Tipos

- Em Ruby não é necessário informar o tipo de uma variável
  - Ela é dinamicamente tipada
- `idade = 27 # Classe Fixnum / Integer`
- `peso = 80.45 # Classe Float`
- `nome = 'Diego' # Classe String`
- Para descobrir o tipo de qualquer variável
  - `puts idade.class`



INSTITUTO FEDERAL

TUDO é Objeto

- Pode-se pedir por `.class` (método) de qualquer coisa (variável, classe, objeto, ...)
  - Em Ruby, TUDO é objeto
    - Mesmo os tipos primitivos
    - Mesmo as classes
    - Inclusive os objetos
    - Até um objeto nulo tem uma classe

```
"Tiririca".class
15.class
55.9.class
true.class
UmaClasse.class
nil.class
```

INSTITUTO FEDERAL

Tipagem Forte

- Se eu não declaro qual o tipo da minha variável, quer dizer que o tipo dela não importa para meu interpretador?
  - MENTIRA
    - Diferentemente do PHP, que é fracamente tipado

No PHP -> OK

```
$idade = 27;
$multiplicador = "2";

$idade * $multiplicador;
```

No Ruby -> ERRO

```
idade = 27
multiplicador = "2"

idade * multiplicador
```

INSTITUTO FEDERAL

Tipagem Dinâmica

- Permite a alteração do tipo de dados de uma variável a qualquer momento

```
idade = 27
idade = "27"
idade = 27.0
idade = true
```

INSTITUTO FEDERAL

Classes Abertas

- Os princípios SOLID pregam:
  - “Entidades devem ser abertas para extensão mas fechadas para modificação.”
- O conceito de classes abertas foge disto
  - Pode-se adicionar novas funcionalidades à uma classe existente a qualquer momento
- O que dá para fazer com isso?
  - `"Marmita".plural`
  - `29.dizer_ola(mensagem)`

INSTITUTO FEDERAL

Convenções de Código

- **CamelCase** para nomes de classes
  - `Aluno`
  - `TipoRegistro`
  - `PeopleController`
- **snake_case** para variáveis, objetos, métodos e nomes de arquivos (em minúsculas)
  - `people_controller.rb`
  - `idade_pessoa = 25`
  - `def calcular_imc(peso, altura)`

INSTITUTO FEDERAL

Parênteses e Ponto-e-vírgula

- Ruby não utiliza ponto-e-vírgula (;) no final das linhas de comando
  - Acostume-se
- Parênteses na chamada de métodos e estruturas de controle são opcionais
  - `calcular_imc peso, altura`
  - `calcular_imc(peso, altura)`

Mesma coisa

INSTITUTO FEDERAL

Caracteres Especiais

- Caracteres especiais permitidos (métodos)
  - ? ← Interrogação
    - Indica que o método vai retornar algo que pode ser verificado em uma condição (pergunta)
  - ! ← Exclamação
    - Indica que o método tem a capacidade de alterar o estado do objeto a que pertence
  - = ← Recebe
    - Serve para definir um método “getter”
- Válido apenas para último caractere do nome

Caracteres Especiais - Exemplos

- `File::exists? "/home/diego/tads15.txt"`
  - O arquivo do parâmetro existe?
- `pessoa.possui_veiculo?`
  - Esta pessoa (objeto) possui um veículo?

- `pessoas = ['Fritz', 'Berlin', 'Franz']`
- `pessoas.sort!`
  - Ordena o vetor “pessoas” definitivamente
  - Diferente de `pessoas.sort`, que só retorna

INSTITUTO FEDERAL

Caracteres Especiais - Exemplos

```
class Pessoa
  def nome= valor
    @nome = valor
  end
end

pessoa = Pessoa.new
pessoa.nome = "Diego"
```

INSTITUTO FEDERAL

Mais Caracteres Especiais

- Outros caracteres especiais podem ser utilizados isoladamente, para fins específicos
  - +
  - -
  - <<
  - !=
  - ==
  - Mais: https://stackoverflow.com/a/10542599/1148768

INSTITUTO FEDERAL

Comentários

- Formato na linha - Cerquilha (#)
  - `idade = 10 # isto é um comentário`
- Formato multilinha - Usar `=begin` e `=end`

```
=begin
   linha não interpretada
   esta também
=end
```

- Princípio Ruby: código auto documentado
  - Se você precisa explicar o que seu código faz, ele não está bem escrito (code smell)

INSTITUTO FEDERAL

Tipos Numéricos

- Integer
  - Tipo para números inteiros
    - `idade = 29`
    - `ano = 2017`
- Float
  - Tipo para números decimais
    - `peso = 10.5`
    - `altura = 1.85`
- Rational
  - Tipo que utiliza frações (sem executar divisão)
    - `pedaco_bolo = Rational('2/3')`

INSTITUTO FEDERAL

Strings

- São objetos da classe String
- Pode-se usar aspas simples ou duplas
  - `nome_completo = "Joana d'Arc"`
  - `mensagem = 'Meu nome é "Diego"!!'`

INSTITUTO FEDERAL

Concatenação / Interpolação

- Podemos usar concatenação para juntar o conteúdo de duas Strings

```
nome = 'Diego'
mensagem = 'Mas já vai embora,'
puts 'Já vai tarde, ' + nome + '!!!'
```

- Mas a forma preferida é com interpolação

```
puts "Já vai tarde, #{nome}"
```

  - Observação: só funciona com aspas duplas

INSTITUTO FEDERAL

Entrada e Saída

- `puts`
  - Função que imprime o que for informado
    - `puts "Mensagem qualquer"`
- `gets`
  - Função que espera por entrada do usuário
    - `nome = gets`
    - `puts "Seu nome é #{nome}"`
  - Retorno do gets vem sujo (espaços, tabs, \n)
    - Para limpar:
      - `nome.strip  # método da classe String`

INSTITUTO FEDERAL

Conversão Rápida de Tipos

- A classe String possui diversos métodos para conversão rápida (`to_`)
  - `texto_lido.to_r # converte para Rational`
  - `texto_lido.to_f # converte para Float`
  - `texto_lido.to_i # converte para Integer`
- O mesmo vale para tipos numéricos
  - `numero = 2.to_f   # converte para Rational`
  - `numero = 2.0.to_i # converte para Integer`
  - `texto = 50.to_s   # converte para String`

INSTITUTO FEDERAL

if

- Estrutura condicional if
  - Parênteses opcionais
  - Bloco termina com palavra reservada end
- Exemplos:

```
nome = "Diego"
idade = 29
if (idade > 18)
    puts nome
end
```

```
peso = 180.0
if peso > 150
    puts 'Gordo'
else
    puts 'Menos gordo'
end
```

INSTITUTO FEDERAL

`if` de uma linha

- Se o bloco do `if` possuir apenas uma linha
  - Pode-se usar uma sintaxe mais enxuta
- Leitura mais natural
  - “Imprima o nome se a idade for maior que 18”

```
nome = "Diego"
idade = 29

puts nome if idade > 18
```

unless

- Traduzindo `unless` = “a menos que”
  - “Faça aquilo, a menos que não deva”
- É a negação do `if` (equivalente ao else)
  - Mas sem precisar fazer o `if`

```
cargo = "Deputado"
nome = "Alfredo"

puts "Bem vindo!" unless cargo == "Deputado"

unless nome == "Diego"
    puts "Seu nome é #{nome}"
end
```

INSTITUTO FEDERAL

Tipo Nulo

- Palavra reservada `nil`
- Em Ruby não temos variáveis de determinado tipo com valor nulo
  - String nula, Pessoa nula, Controller nulo, ...
- O `nil` indica que nada existe ali
  - Instância de `NilClass`
- Exemplos
  - `caixa = nil`
  - `idade = nil`

Método nil?

- Todo objeto possui o método nil?
  - Não esquecer da interrogação (?)
  - Retorna verdadeiro caso aquele objeto seja uma instância de NilClass (== nil)

```
#mostra a saída
nome = "Diego"
puts "Seja bem-vindo #{nome}" if not nome.nil?

#não mostra a saída
nome = nil
puts "Seja bem-vindo #{nome}" unless nome.nil?
```

Melhorando o Código

- Usando uma variável nula (do tipo `nil`, (instância de `NilClass`) em uma condição, ela se comporta como um valor booleano
  - Exemplo:

```
#mostra a saída porque nome == true
nome = "Diego"
puts "Seja bem-vindo #{nome}" if nome

#não mostra a saída porque nome == false
nome = nil
puts "Seja bem-vindo #{nome}" if nome
```

INSTITUTO FEDERAL

Iterações Simples

- Existem diversas formas de iterar (executar repetidamente) determinado trecho de código Ruby
- Formas mais comuns (e semelhantes a outras linguagens)
  - `for`
  - `while`
  - `until`
- Todos podem ser quebrados por um `break`

INSTITUTO FEDERAL

for (for each)

- Usado quando desejamos iterar de um número a outro usando uma variável
- Sintaxe (parênteses opcionais)

```
for variavel in (inicio..fim) #inclusive
    # Corpo a ser repetido
end
```

- Exemplo

```
for numero in 1..100
    puts "Numero: #{numero}"
end
```

INSTITUTO FEDERAL

while

- Sem mais delongas, igual a outras linguagens
- Exemplo

```
numero = 0
while numero <= 100
  puts "Numero: #{numero}"
  numero += 1
end
```

INSTITUTO FEDERAL

until

- Tradução: até que
  - Faça tal coisa até que a condição seja válida
  - Criada pensando na legibilidade do código
- Contrário do `while`
  - Pára quando a condição é verdadeira
- Exemplo

```
numero = 0
until numero == 100
    puts "Numero: #{numero}"
    numero += 1
end
```

INSTITUTO FEDERAL

.each (block)

- **Block** (bloco) é uma técnica que permite a delegação (terceirização) de trechos de código
  - Veremos melhor mais à frente
- Existe uma forma de iteração que usa blocks
  - Largamente utilizada, inclusive em Rails
- Exemplo:

```
pessoas = ['Fritz', 'Franz', 'Berlin']
pessoas.each do |pessoa|
    puts "Nome do jovem: #{pessoa}"
end
```

INSTITUTO FEDERAL

Funções / Métodos

- Utiliza-se a palavra reservada def
  - Se return não for especificado
    - Retorna a última linha executada
- Sintaxe

```
def nome_da_funcao(param1, param2)
    #corpo da função
end
```

- Exemplo

Parênteses opcionais aqui também

```
def calcula_imc(peso, altura)
    peso / (altura * altura) # vai retornar isso
end
```

INSTITUTO FEDERAL

Funções - Invocação

- Lembrar que parênteses são opcionais
  - `imc = calcular_imc(81, 1.85)`
  - `imc = calcular_imc 81, 1.85`

  - `dar_boas_vindas()`
  - `dar_boas_vindas`

  - `nome = ler_do_teclado()`
  - `nome = ler_do_teclado`

INSTITUTO FEDERAL

Arrays

- Arrays em Ruby têm tamanho dinâmico
  - Classe `Array`
- Declaração mais simples: `[]`
  - `sucessos_na_minha_vida = []`
  - `numeros = [10, 20, 30]`
- Indexados a partir de 0 (zero)
  - `puts numeros[0]`
  - `puts numeros[1]`

Array

- Como é um objeto da classe `Array`
  - Temos muitos métodos utilitários
- Exemplos

```
numeros = [10, 20, 30]
puts numeros.first
puts numeros.last
numeros.clear

#adiciona um item ao final do vetor
x << 50          # "<<" é o nome do método
x << "chuchu"    # o tipo não importa
```

INSTITUTO FEDERAL

Percorrendo um Array

- Podemos usar o `for` visto
  - Que na verdade é um `for each`
- Sintaxe

```
for item in vetor do
    #corpo do for
end
```

- Exemplo

```
for palavra in palavras do
   puts "Sua palavra: #{palavra}"
end
```

INSTITUTO FEDERAL

Percorrendo um Array

- Ou podemos usar método `.each`
  - Utilização de blocks
- Sintaxe

```
vetor.each do |item|
    #corpo do for
end
```

- Exemplo

```
palavras.each do |palavra|
   puts "Sua palavra: #{palavra}"
end
```

INSTITUTO FEDERAL

Symbol

- Ruby possui um tipo de dados especial chamado `Symbol` (Símbolo)
  - Sua sintaxe utiliza dois pontos (`:`) no nome
    - `:nome` `:ruby` `:java`
- Tem comportamento equivalente a uma String estática (inalterável)
  - Ocupa menos memória e evita erros
  - Pode ser facilmente convertido para String
    - `:nome.to_s # retorna "nome"`

INSTITUTO FEDERAL

Symbol - Exemplo

```
def voce_gosta_de_ruby?(resposta)
    if resposta == :sim
        puts "Você é bacana"
    else
        puts "Bobão!"
    end
end


voce_gosta_de_ruby? :sim
```

INSTITUTO FEDERAL

Hash

- Parecido com array
  - Mas indexados por objetos de qualquer classe
- Estrutura lembra o Object do JavaScript
- Utilizar pares
  - `{chave => valor}`
- Instanciação → `{}` ou `Hash.new`
- Exemplo
  - `idades = {'Diego' => 29, 'Zé' => 55}`

INSTITUTO FEDERAL

Hash - Exemplo

```
h = {
    'cachorro' => 'canino',
    'gato' => 'felino',
    'cavalo' => 'cavalino',
    12 => 'número doze'}

puts h.length  # 4
puts h['cachorro']  # 'canino'
puts h
puts h[12]
```


https://guru-sp.github.io/tutorial_ruby/hashes-ruby.html

INSTITUTO FEDERAL

Hash - Exemplo

- É muito comum indexar Hashes com Symbols

```
pessoa = Hash.new
pessoa[:nome] = 'Diego'
pessoa[:sobrenome] = 'Maradona'
pessoa[:idioma] = 'Espanhol'

puts pessoa[:sobrenome] # Maradona
```

INSTITUTO FEDERAL

Hash - Exemplo

- Mesma forma de fazer o Hash anterior

```
pessoa = {:nome => 'Diego', :sobrenome
=> 'Maradona', :idioma => 'Espanhol'}
```

- Forma abreviada

```
pessoa = {nome:  'Diego', sobrenome:
'Maradona', idioma: 'Espanhol'}
```

- Estes formatos (com Symbol) são muito utilizados pelo framework Ruby on Rails

INSTITUTO FEDERAL

Hash

- Armazenando arrays

```
pessoas = {
    chatos: ['Quico', 'Mr. Burns', 'Faustão'],
    legais: ['Silvio Santos', 'Homer', 'Chaves']
}

puts pessoas[:chatos].first
puts pessoas[:legais].last
pessoas[:legais] << 'Oilman'
```