

Ruby on Rails 5.x

Do início ao fim!

Módulo 02

Ruby Básico

Ruby e RVM



Ruby

Linguagem de Programação



RVM

Ruby Version Manager



RVM

<https://rvm.io/>

RVM

Comandos úteis...

Comandos úteis...

- Listar todas as versões disponíveis no repositório
 - `rvm list known`
- Atualiza a lista
 - `rvm get head`
- Lista as versões instaladas localmente
 - `rvm list`
- Instala uma versão escolhida
 - `rvm install x.x.x`

Comandos úteis...

- Instala uma versão escolhida e a torna padrão
 - `rvm install x.x.x --default`
- Usa uma versão específica do ruby
 - `rvm use x.x.x`

Ruby, IRB e Pry



**Como executar um
arquivo ruby?**

Como executar um arquivo ruby?

- Após escrever o algoritmo em arquivo, rode no terminal:
 - `ruby meu_arquivo.rb`



Conhecendo o IRB

Conhecendo o IRB

Interactive Ruby Shell

Conhecendo o IRB

https://en.wikipedia.org/wiki/Interactive_Ruby_Shell

Conhecendo o IRB

- No prompt digite **"irb"**
- Dentro do IRB você pode digitar qualquer instrução Ruby para ser interpretada. Tente:
 - `1+2`
 - `"curso de ruby on rails".reverse`
 - `a = {curso: "rails", versao: "5.x"}`

Conhecendo o Pry

Conhecendo o Pry

<http://pryrepl.org/>

Conhecendo o Pry

- Instale o pry
 - `gem install pry`
- Inicie o pry rodando "pry"
- Use-o como um REPL normal
 - `a = {curso: "rails", versao: "5.x"}`

Conhecendo o Pry

- Tornando **pry** padrão ao chamar o **irb**

Crie o arquivo .irbrc na home do usuário

- `touch ~/.irbrc`

Dentro do arquivo .irbrc

- `require 'rubygems'`
- `require 'pry'`
- `Pry.start`
- `exit`

Variáveis e Versões do Ruby



**Como declarar uma
variável em Ruby?**

Como declarar uma variável em Ruby?

- Variáveis em Ruby são declaradas apenas "usando-a".
- O Ruby infere o tipo da variável no momento da "declaração".
 - `x = 1`
 - `y = 2.3`
 - `z = "Rails 5.x"`
- Para verificar o tipo da variável, use o método `.class`

Versões

2.3 VS 2.4+

Versões 2.3 vs 2.4+

- **2.3**
 - Fixnum representa os inteiros
 - Bignum representa os números "inteiros gigantes"
- **2.4+**
 - Integer representa todos os números inteiros

Saída e Entrada padrão, `\n`, `Chomp` e Coerção



Saída Padrão

STDOUT

Saída padrão

- A saída padrão STDOUT é representado pela "tela".
- No Ruby usamos o **puts** para imprimir algo na tela.
 - `puts "Curso de Rails 5.x"`

Entrada Padrão

STDIN

Entrada padrão

- A entrada padrão STDIN é representado pelo teclado.
- No Ruby usamos o **gets** para "ler" algo do teclado.
 - `a = gets`

\n e .chomp

`\n` e `.chomp`

- O Código de formatação `\n` indica "**new line**", ou seja, adiciona uma nova linha, ou, como dizemos, "**quebra uma linha**".
- O `.chomp` é um método que remove o código de formatação `\n` do elemento ao qual foi aplicado.

The background features a complex arrangement of overlapping geometric shapes in various shades of gray. A faint, light-gray silhouette of a human face is visible on the right side, partially obscured by the geometric patterns. The overall aesthetic is modern and minimalist.

Coerção

cast ou casting

Coerção

- Coerção, cast ou casting é o procedimento que fazemos para "transformar" o conteúdo de uma variável/entrada em outro tipo.
- Nesse caso, não modificamos a variável, apenas "interpretamos" ela como um tipo que queremos.
- `gets.to_i`
- `x = "2.5"`
- `x.to_f`

The background features a light gray abstract design with overlapping geometric shapes, including triangles and polygons. A faint, stylized globe is visible in the upper right corner. The text is centered on the left side of the image.

Estruturas Condicionais

Estruturas condicionais

if, unless, case

Estruturas condicionais...

- if

- a. `x = 1`

- b. `if x > 2`

- c. `puts "x é maior que 2"`

- d. `end`

Estruturas condicionais...

- **unless**

```
a. x = 1
b. unless x >= 2
c.     puts "x é menor que 2"
d. else
e.     puts "x é maior que 2"
f. end
```

Estruturas condicionais...

- **case**

```
a. idade = 5
b. case idade
c.   when 0 .. 2
d.     puts "bebê"
e.   when 3 .. 12
f.     puts "criança"
g.   when 13 .. 18
h.     puts "adolescente"
i.   else
j.     puts "adulto"
k.   end
```

Estruturas condicional ternária

<condição> ? <verdadeiro> : <falso>

Operadores Relacionais e Aritméticos

Relacionais

$>$, $>=$, $<$, $<=$, $==$, $!=$

Aritméticos

$+$, $-$, $*$, $/$, $**$, $\%$



Estruturas de Repetição

Estruturas de Repetição

while, each

Estruturas de Repetição

- while

a. `i = 0`

b. `num = 5`

c.

d. `while i < num do`

e. `puts "Contando... " + i.to_s`

f. `i += 1`

g. `end`

Estruturas de Repetição

- each

```
a. (0..5).each do |i|
```

```
b.     puts "O valor lido foi: " + i.to_s
```

```
c. end
```

Arrays / Vetores

Vetores / Array

- Você pode declarar/usar de duas formas:

a. `v = [15, 62, 73, 48]`

- **OU**

a. `v = Array.new`

b. `v.push(15)`

c. `v.push(62)`

Vetores / Array

- Para acessar, use o índice

a. `puts v[0]`

V	15	62	73	48
	[0]	[1]	[2]	[3]

Vetores / Array

- No Ruby, os arrays são dinâmicos e aceitam armazenar tipos diferentes dados

a. `v = ["curso", 62, 1.4]`

b. `v.push("hello")`

Vetores / Array

- Arrays podem ser aninhados:
 - `v = [[11,12,13], [21,22,23], [31,32,33]]`
- Podemos usar o `each` para iterar
 - `v.each do |externo|`
 - `externo.each do |interno|`
 - `puts interno`
 - `end`
 - `end`

Hashes



Hash

Uma lista do tipo **chave => valor**

Relembrando

Em um vetor usamos o índice (chave fixa) para acessar o valor armazenado...

V	15	62	73	48
	[0]	[1]	[2]	[3]

Hash

Em um hash você determina qual é a “chave” para acessar o valor

H	15	“rails”	1997	32
	“x”	“curso”	“ano”	1

Hash

- Você pode criar hashes de duas formas
 - **Tradicional:**
 - `h = {"x" => 15, "curso" => "rails"}`
 - **1.9+**
 - `h = {"x": 15, "curso": "rails"}`
- Para acessar os elementos, use os []
 - `h["curso"]`

Strings, Concatenação e Interpolação de Variáveis

Strings



Strings

- Strings são determinadas por usar as aspas duplas ou simples

a. `x = "Curso de Rails"`

b. `y = 'Curso de Ruby'`

Concatenação de Strings

Concatenação de Strings

- +

```
a. x = "rails"  
b. y = "curso " + x  
c. puts y
```

- << (shovel)

```
a. x = "rails"  
b. y = "curso " << x  
c. puts y
```

Concatenação de Strings

- Diferença entre + e <<

```
a. x = "curso"
b. puts x.object_id
c. x = x + "rails"
d. puts x
e. puts x.object_id
f. #####
g. q = "curso de "
h. puts q.object_id
i. q << "rails"
j. puts q
k. puts q.object_id
```

- O + gera um novo objeto sempre que usado

Interpolação de Variáveis

Interpolação de Variáveis

- Use a combinação `# { }` para interpolar strings com variáveis ou código ruby

a. `x = "Jackson"`

b. `puts "Seu nome é #{x}"`

- Apenas strings criadas com aspas duplas são interpoláveis

Símbolos / Symbols

Símbolos / Symbols

- Símbolos são “strings imutáveis”

- a. `puts "jackson".object_id`
- b. `puts "jackson".object_id`
- c. `puts "jackson".object_id`
- d. `#####`
- e. `puts :jackson.object_id`
- f. `puts :jackson.object_id`
- g. `puts :jackson.object_id`

Símbolos / Symbols

- Símbolos são muito usados em situações onde precisamos de um identificador pois eles garantem que seu uso não implicará na criação de novos objetos sempre que usados.
- Hashes adoram símbolos:
 - a. `h = { :curso => "Rails" }`
 - b. `h = { curso: "Rails" }`

Sobre Parênteses e Constantes

Parênteses

Parênteses

- No Ruby, o uso de parênteses é opcional

a. `puts("Curso Rails")`

b. `puts "Curso Rails"`

Constantes

Constantes

- São usadas para representar valores

a. `NOME = "Jackson"`

b. `PI = 3.14`

c. `puts NOME`

d. `puts PI`

Tipos Primitivos vs Complexos

Tipos Primitivos

Tipos básicos da computação

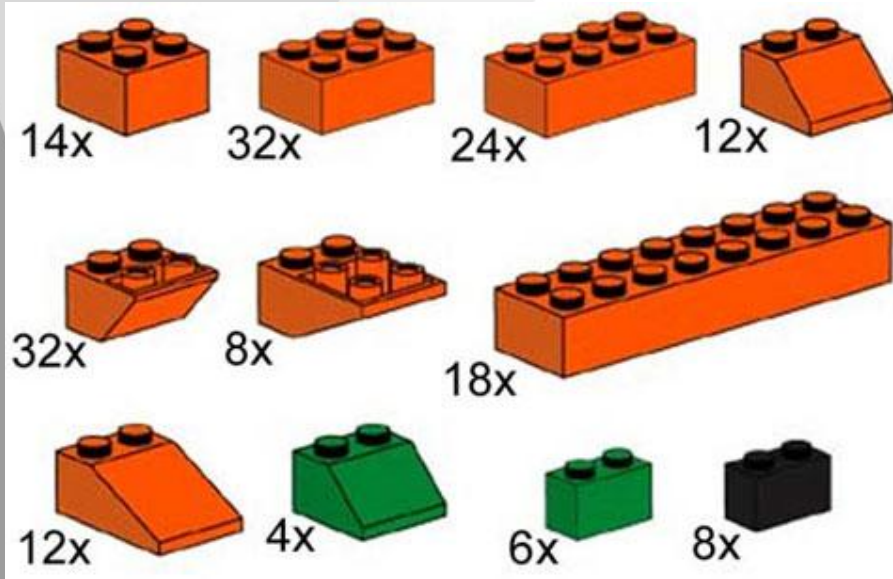
Tipos primitivos

- Inteiro
 - `x = 1`
- Real / Float
 - `y = 3.45`
- Caracteres / String
 - `z = 'abc'`
- Lógico / Booleano
 - `w = true`

Tipos Complexos

Tipos complexos

- Usamos os tipos primitivos para criar tipos complexos



Primitivos



Complexos

Tipos complexos

- Podemos dizer que os tipos complexos são as **classes/objetos**



Complexos

Tipos complexos

- Exemplo
 - A classe **Date**

dd/mm/yyyy

22/12/2009

i c i c i



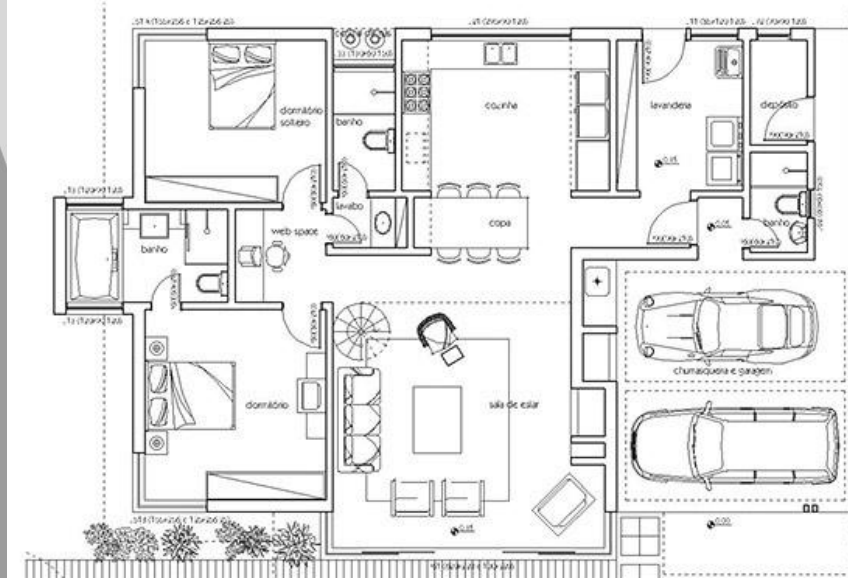
Classes vs Objetos

Classes vs Objetos

“formas” vs “objeto pronto”

Classes vs Objetos

Forma

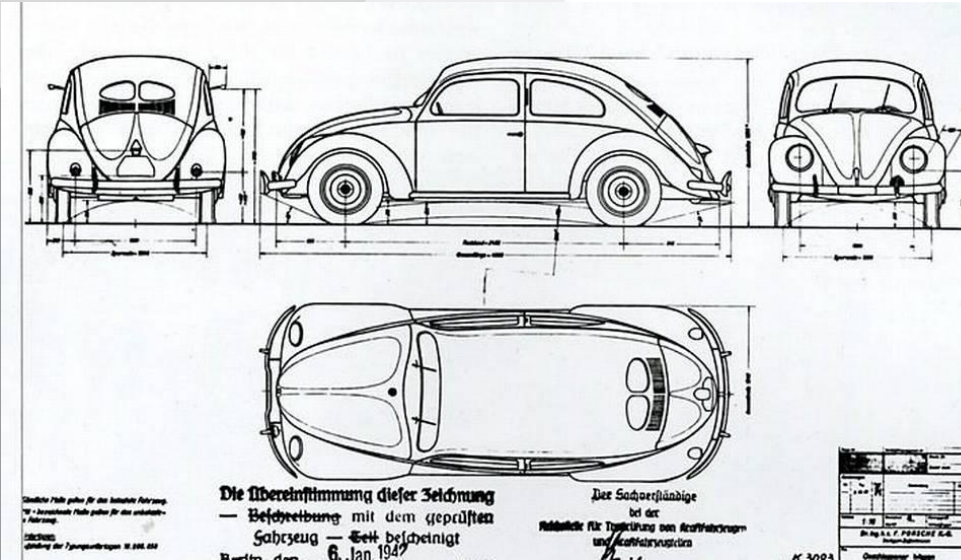


Objeto Pronto



Classes vs Objetos

Forma



Objeto Pronto



Classes vs Objetos

- As classes são a maneira que temos de informar como queremos que nosso objeto funcione!
- Ao criarmos uma classe podemos especificar os métodos e os atributos que os objetos possuirão.
- Os **métodos** são as **ações**.
- Os **atributos** são as **características**

Métodos e Ações

- Quais os métodos/ações?
- Quais os atributos/características?



Métodos e Ações

- Quais os métodos/ações? (*acelera, freia, liga o farol*)
- Quais os atributos/características? (*é da cor cinza, possui pneus aro 13"*)





Tudo no Ruby é objeto!

`.class`

Identificador único do objeto

.object_id

Qual a vantagem de tudo ser objeto no Ruby?

Você “ganha” automaticamente várias ações/métodos em seus objetos.



**Conheça todos os
métodos do objeto
usando TAB!**

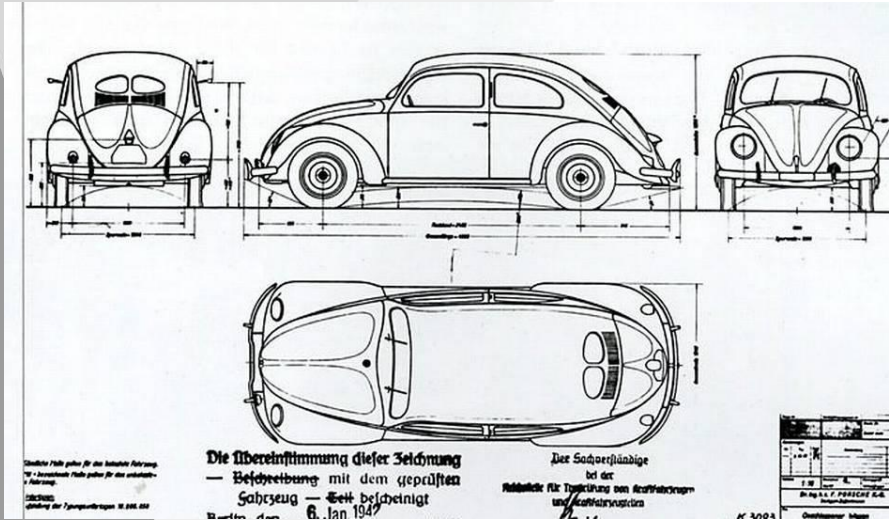
Criando Classes e Métodos

Lembre-se!

Uma classe instanciada
é um objeto!

Lembre-se!

- Uma classe instanciada é um objeto



Lembre-se!

- A classe instanciada é um objeto
 - `# por inferência`
 - `a = [61, 92, 37]`
 -
 - `# declaração explícita`
 - `b = Array.new`
 - `b.push(53)`

The background features a light gray abstract design with various geometric shapes, including triangles and polygons. A faint, light gray silhouette of a car is visible on the right side. The text is centered in a bold, dark red font.

Como criar uma classe?

Como criar uma classe?

- Crie um arquivo `pessoa.rb`
 - `class Pessoa`
 - `...`
 - `end`
- A classe sempre começa com uma letra maiúscula (capitulada)
- Nomes compostos devem capitalizar o início de cada palavra. Ex: `PessoaFisica`

Como criar uma classe?

- Use o `irb` para testar
- Use o `require_relative`
 - `require_relative "arquivo.rb"`
 - `p = Pessoa.new`
- ps: O nome do arquivo não é obrigado ser o mesmo da classe!

Métodos



**Como criar métodos
(ações)?**

Como criar métodos (ações)?

- Dentro da classe, use o `def <nome da acao> / end`
 - `def falar`
 - `"Olá, pessoal!"`
 - `end`

Parâmetros e Initialize

Parâmetros

É a forma de passar dados para dentro do método

Parâmetros

- `def falar(texto)`
- `"Olá!, #{texto}"`
- `end`

Parâmetros

Você pode usar um valor padrão...

- `def falar(texto = "Olá, tudo bem?")`
- `texto`
- `end`

Parâmetros

Você pode usar mais de um parâmetro...

- `def falar(texto = "Olá!", texto2 = "Hello!")`
- `"#{texto} - #{texto2}"`
- `end`



O método *initialize*

O método *initialize*

- O método *initialize* é um método especial que serve para indicarmos o que a classe deve fazer ao ser instanciada/inicializada.
 - `def initialize`
 - `puts "inicializando..."`
 - `end`

O método initialize

- Você também pode usar parâmetros na inicialização

```
o def initialize(cont = 5)
o   cont.times do |i|
o     puts "Contando.. #{i}"
o   end
o end
```

Self, Variáveis de Instância e Accessors



Self

Self

- A grosso modo o `self` é o próprio objeto, ou seja, o objeto instanciado.
 - `def meu_id`
 - `"Meu ID é o: #{self.object_id}"`
 - `end`

Self

- Você também pode reabrir classes no Ruby e usar o `self` veja esse exemplo:
 - `class String`
 - `def inverter`
 - `self.reverse`
 - `end`
 - `end`

Variáveis de Instância

Variáveis de Instância

- Variáveis de instâncias são as variáveis que existem apenas na instância do objeto (em todo objeto), ou seja, cada objeto possui seus próprios valores em tais variáveis
- As variáveis de instância são precedidas de um @
 - `def initialize(nome_fornecido = "indigente")`
 - `@nome = nome_fornecido`
 - `end`
 - `def imprimir_nome`
 - `@nome`
 - `end`

The background features a complex geometric design. On the left, there are several overlapping triangles in shades of gray. A large, light gray shape resembling a gear or a stylized letter 'A' dominates the center and right side. This shape has several white, pill-shaped cutouts. The word "Accessors" is centered over the gear-like shape.

Accessors

Accessors

- Os *accessors* servem como atalhos para declaração de atributos de uma classe. Veja o exemplo:
 - `attr_accessor :nome`
- A simples declaração acima te dá um “getter” e um “setter” para **nome** na classe em questão.
 - `x = Pessoa.new`
 - `x.nome = "Jackson"`
 - `x.nome`

Classes e Herança

Representando uma classe visualmente

Notação UML

Representando uma classe visualmente

- Classe Pessoa representada usando UML

```
class Pessoa
  attr_accessor :nome, :email

  def falar(texto)
    texto
  end

  def gritar(texto)
    "#{texto}!!!"
  end
end
```

Pessoa
nome email
falar(texto) gritar(texto)

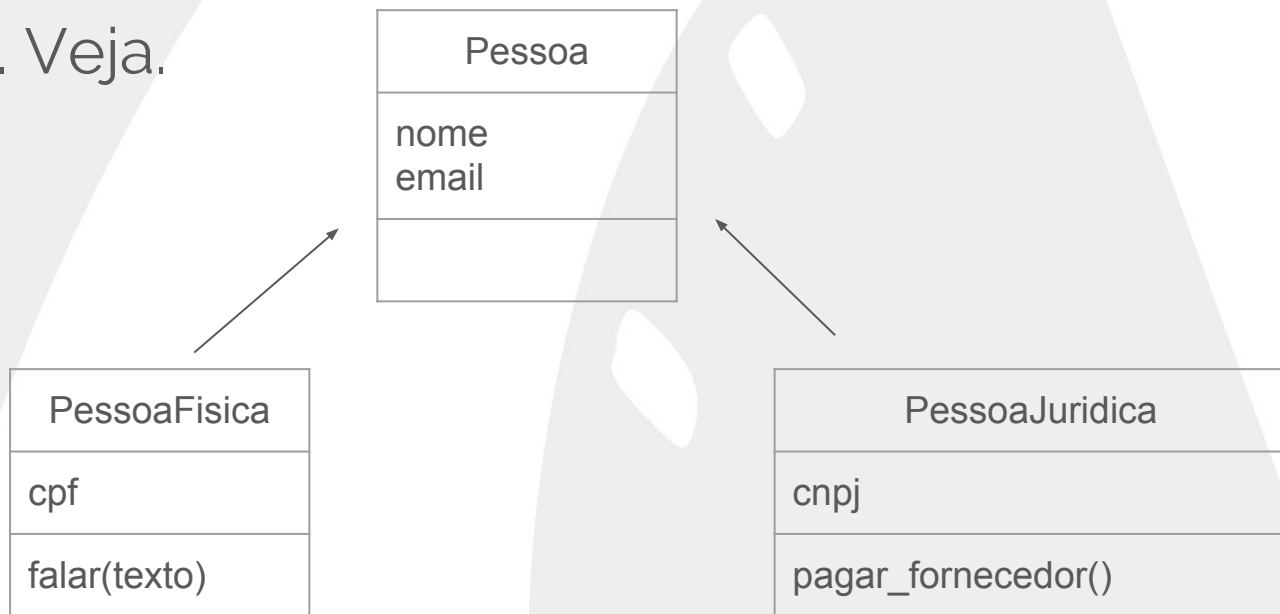
Herança entre Classes

Herança entre Classes

- Imagine a seguinte situação:
 - Pessoa
 - Pessoa Física
 - Pessoa Jurídica
- Você deve concordar que tanto a pessoa física como a jurídica “herdam” características e ações da “pessoa genérica”. Por exemplo: Ambas possuem nome, endereço, etc.

Herança entre Classes

- Podemos representar essa situação através de Herança. Veja.



Herança entre Classes

- Convertendo em código seria:

```
class Pessoa
  attr_accessor :nome, :email
end
```

```
class PessoaFisica < Pessoa
  attr_accessor :cpf
```

```
  def falar(texto)
    texto
  end
end
```

```
class PessoaJuridica < Pessoa
  attr_accessor :cnpj
```

```
  def pagar_fornecedor
    "pagando fornecedor..."
  end
end
```

Herança entre Classes

- Ou seja, usa-se o menor “<” para indicar a herança.
- No Ruby não existe herança múltipla, ou seja, não é possível herdar de várias classes ao mesmo tempo.

Métodos de Instância e de Classe

Antes...

O puts e p2 da aula anterior...

Ainda Antes...

<https://www.videosdeti.com.br/>

- Fundamentos de Programação
- Orientação a Objetos com Ruby
- Banco de Dados SQL

Métodos de Instância

Métodos de Instância

- São os métodos que só podem ser invocados a partir de um objeto, ou seja, uma classe instanciada.

```
class Pessoa
  attr_accessor :nome, :email

  def falar(texto)
    texto
  end
end
```

Métodos de Classe

Métodos de Classe

- São métodos que podem ser executados a partir da própria classe, ou seja, não é necessário instanciar um objeto.

```
class Pessoa
  attr_accessor :nome, :email

  def falar(texto)
    texto
  end

  def self.gritar(texto)
    "#{texto}!!!"
  end
end
```

```
Pessoa.gritar("Hello")

#=> Hello!!!
```

Módulos e Mixins



Módulos

Módulos

- Módulos Ruby são similares a classes em relação ao fato de que também armazenam uma coleção de **métodos**, **constantes** e outras definições de **módulos e classes**.

Módulos

- Entretanto, diferente das classes, você não pode criar objetos baseados em módulos nem pode criar módulos que herdaram desse módulo; ao invés disso, você especifica qual funcionalidade de um módulo específico você deseja adicionar a uma classe ou a um objeto específico.

Módulos

- Módulos permanecem sozinhos; não há hierarquia de módulos ou herança. Módulos são um bom lugar para armazenar constantes em um local centralizado.

The background features a light gray abstract design with overlapping geometric shapes, including triangles and polygons. A faint, stylized globe is visible in the background, centered behind the text. The text is in a bold, dark red font.

Principais objetivos dos Módulos

Módulos

- Primeiro eles agem como *namespace*, permitindo que você defina métodos cujos nomes não irão colidir com aqueles definidos em outras partes de um programa.

Módulos e Mixins

- Em segundo lugar, permitem que você compartilhe funcionalidade entre classes – se uma classe “mistura” (**mixes in**) um módulo (isto é, o inclui), todos os métodos de instância do módulo se tornam disponíveis como se tivessem sido definidos na classe.

Módulos e Mixins

- Exemplo para Constantes

```
# pagamento.rb
```

```
module Pagamento  
  PI = 3.14  
end
```

```
# app.rb
```

```
require_relative 'pagamento'  
  
include Pagamento  
  
puts Pagamento::PI  
puts PI
```

Módulos e Mixins

- Exemplo para Métodos

```
# pagamento.rb

module Pagamento
  def pagar(bandeira, numero, valor)
    "Pagando com o cartão #{bandeira} Número #{numero}, o valor de
    R$#{valor}..."
  end
end
```

Módulos e Mixins

- Exemplo para Métodos

```
# app.rb
require_relative 'pagamento'
include Pagamento

puts "Digite a bandeira do cartão:"
b = gets.chomp
puts "Digite a número do cartão:"
n = gets.chomp
puts "Digite a valor da compra:"
v = gets.chomp

puts pagar(b, n, v)
puts Pagamento::pagar(b, n, v)
```

Módulos e Mixins

- Exemplo para Classes

```
# pagamento.rb

module Pagamento
  class Visa
    def pagando
      "pagando..."
    end
  end
end
```

```
# app.rb

require_relative 'pagamento'

include Pagamento

p = Pagamento::Visa.new
puts p.pagando
```


Módulos e Mixins

- Exemplo para Módulos dentro de Módulos

```
# pagamento.rb

module Pagamento
  module Master
    def pagando
      "pagando..."
    end
  end
end
```

```
# app.rb

require_relative 'pagamento'

include Pagamento::Master

puts Pagamento::Master::pagando
```

The background features a light gray abstract design with several overlapping geometric shapes, including triangles and polygons. Scattered throughout these shapes are several white, irregularly shaped elements that resemble gemstones or crystals. The overall aesthetic is clean and modern.

Um pouco mais sobre Gems

Gems

Gems são bibliotecas ou conjuntos de arquivos Ruby reutilizáveis, etiquetados com um nome e uma versão.



Pesquisando...

<http://rubygems.org>

The background features a complex arrangement of overlapping geometric shapes in various shades of gray. Several white, gemstone-like shapes are scattered throughout, including two square-cut stones on the left and several irregular, organic shapes on the right. The overall composition is abstract and modern.

Listando as gems

Listando as gems

- Gems instaladas localmente (perceba as versões)
 - `gem list`
- Pesquisa aproximada (localmente)
 - `gem list <nome da gem>`
- Pesquisa aproximada (remotamente)
 - `gem list <nome da gem> --remote`
- Pesquisa aproximada (remotamente para todas as versões)
 - `gem list <nome da gem> --remote --all`



Instalando

Instalando

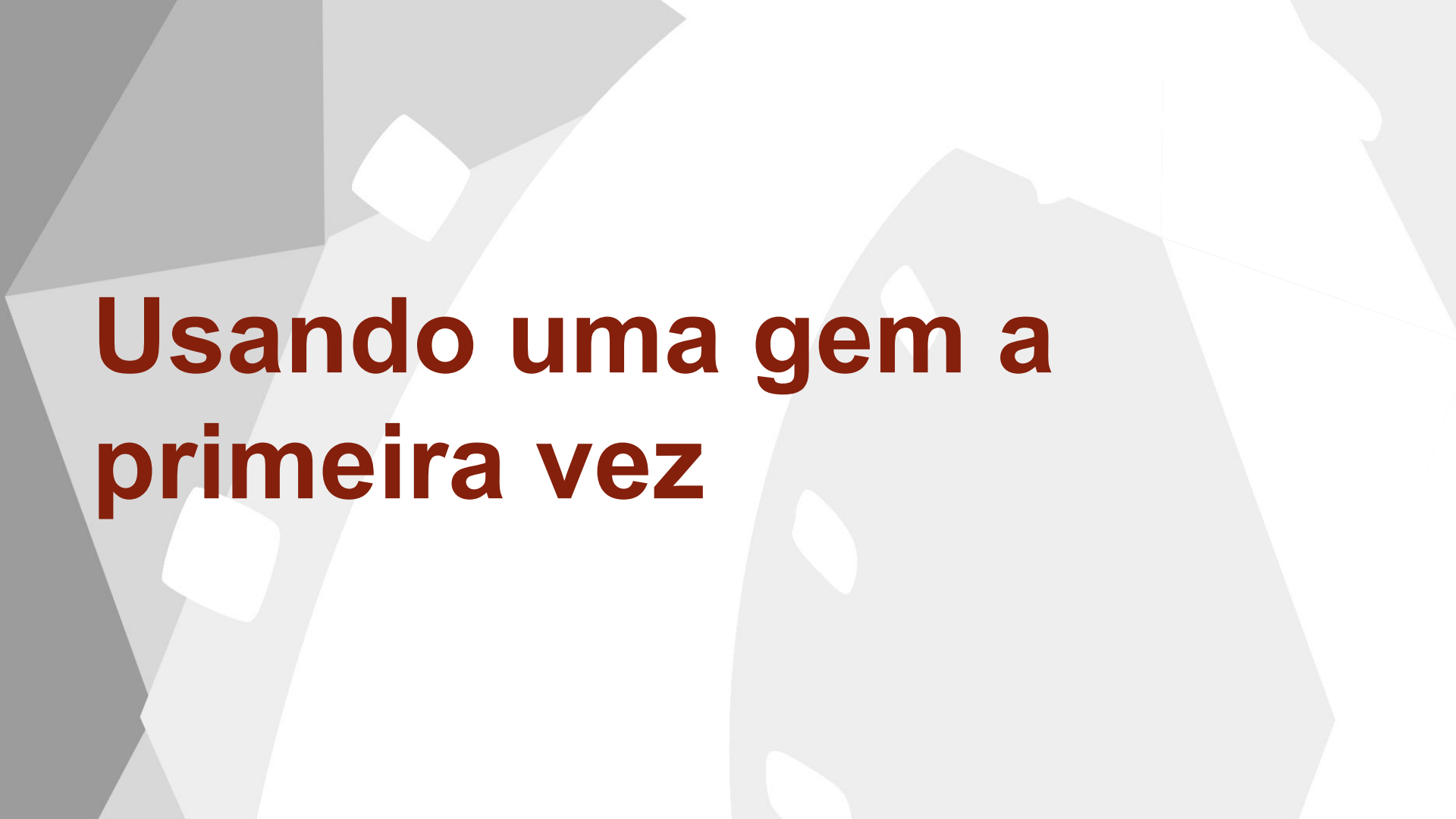
- Instalação básica de uma gem
 - `gem install <nome da gem>`
- Instala uma gem em uma versão específica
 - `gem install <nome da gem> -v <versão x.x.x>`



Removendo

Removendo

- Remove uma gem
 - `gem uninstall <nome da gem>`
- Remove versões antigas da gem
 - `gem cleanup`
- Remove versões antigas de uma determinada gem
 - `gem cleanup <nome da gem>`
- Verifica versões que serão apagadas
 - `gem cleanup -d`



Usando uma gem a primeira vez



Siga os passos...

Siga os passos...

- Instale a gem
 - `gem install cpf_utils`
- Importe a gem no seu projeto/arquivo/irb
 - `require 'cpf_utils'`
- Use a gem
 - `CpfUtils.cpf`



O Bundler

O que são dependências?

Algumas gems precisam de outras gems para funcionar corretamente.



O Bundler

<http://bundler.io/>

O Bundler

- O bundler é uma gem!
- Como funciona?
 - Crie um arquivo **Gemfile**
 - Gemfile
 - Adicione um repositório
 - `source 'https://rubygems.org'`
 - Adicione as gems que deseja instalar
 - `gem 'lerolero'`
 - `gem 'cpf_utils'`
 - `gem 'faker'`

O Bundler

- Após adicionar as gems salve, saia do arquivo e execute no terminal o bundle
 - `bundle install`
- Nesse momento será gerado um arquivo **Gemfile.lock** contendo informações sobre todas as gems que seu projeto usa.
- Isso vai te dar a possibilidade de junto aos seus projetos indicar quais gems foram usadas.

The background features a complex arrangement of overlapping geometric shapes in various shades of gray. Several white, faceted gemstone-like shapes are scattered throughout, some appearing as if they are part of the larger gray structures. The overall aesthetic is modern and minimalist.

Versionamento de Gems

The background is a light gray with several overlapping geometric shapes in various shades of gray. A white film strip with three sprocket holes is visible on the left side, curving upwards. The text "Antes..." is centered in a bold, dark red font.

Antes...

The background features a light gray abstract design with overlapping geometric shapes, including triangles and polygons. A faint, stylized globe is visible in the upper right corner. The text is centered in a bold, dark red font.

Sistema comum de versionamento...

Sistema comum de versionamento...

- X.Y.Z (Major.Minor.Patch)
 - `gem "cpf_utils", "1.0.0"`

Basicamente o **Patch** é para correções, **Minor** para alterações pequenas e **Major** quando as alterações são grandes, muitas vezes impactando na forma de usar a gem.



Versionamento de Gems

Veja esses exemplos...

Versionamento de Gems...

```
gem 'lerolero', '1.0.1'
```

```
gem 'cpf_utils', '>=1.0.0'
```

```
gem 'faker', '~>1.6.0'
```


Versão Exata

gem 'lerolero', '1.0.1'

Versão Igual ou maior que...

gem 'cpf_utils', '>=1.0.0'

Versão 'parcial' atual...

gem 'faker', '~>1.6'

< 2.0

Versão 'parcial' atual...

gem 'faker', '~>1.0.2'

< 1.1

The background features a light gray abstract design with several overlapping geometric shapes, including triangles and polygons. Scattered throughout are white, irregular shapes that resemble gemstones or crystals. The text is centered over this background.

Conhecendo algumas gems

Conhecendo algumas gems

- Lero-Lero Generator
 - https://github.com/jacksonpires/lerolero_generator
- CPF Utils
 - https://github.com/jacksonpires/cpf_utils
- Documentos BR
 - https://github.com/jacksonpires/documentos_br
- TTY Toolbox
 - <http://piotrmurach.github.io/tty/>