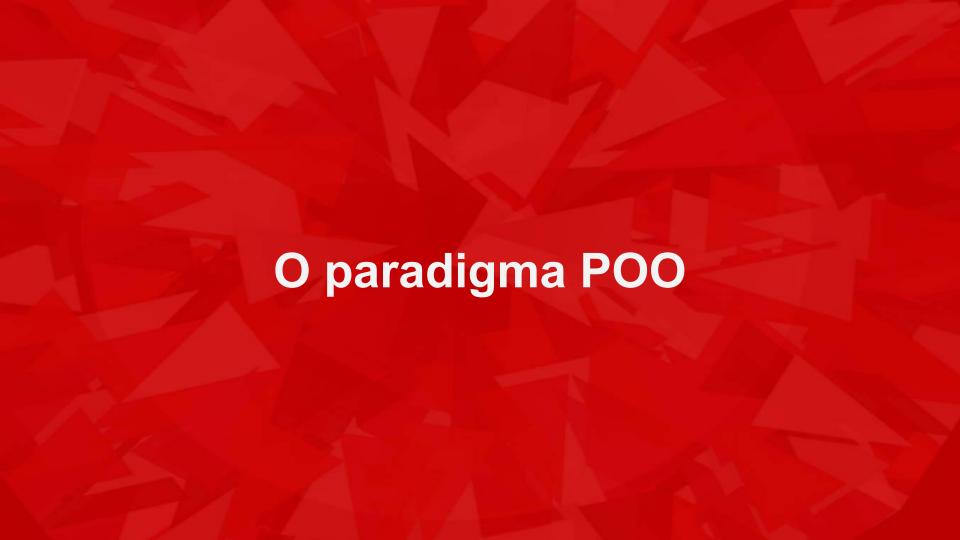
# Introdução ao Ruby on Rails

by Jackson Pires

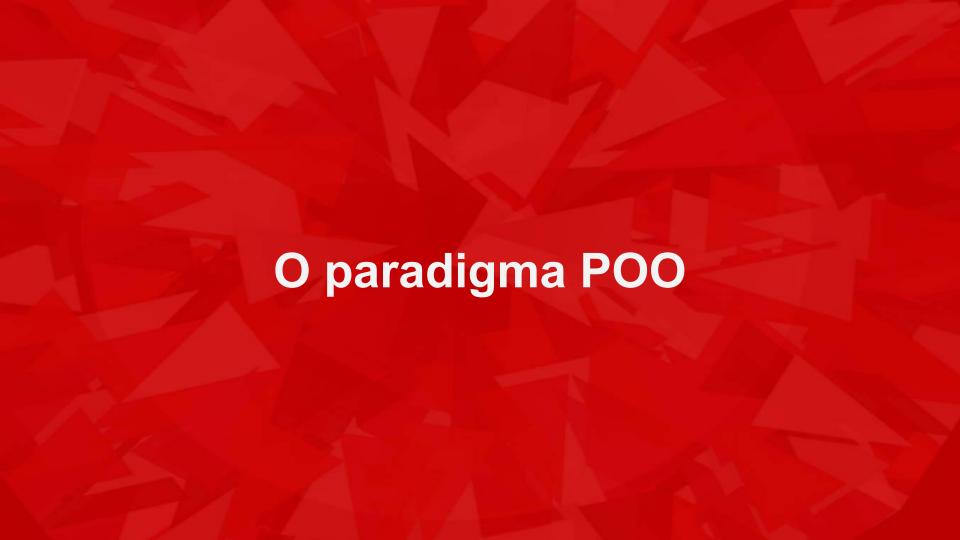






### **Errata**

- if(5>7); end
- if 5>7; end
- if (5>7) then; end
- if 5>7 then; end



- "Paradigma é um conjunto de regras que estabelecem fronteiras e descrevem como resolver os problemas dentro destas fronteiras."

- "Paradigma é um conjunto de regras que estabelecem fronteiras e descrevem como resolver os problemas dentro destas fronteiras."
- "Os paradigmas influenciam nossa percepção; ajudam-nos a organizar e a coordenar a maneira como olhamos para o mundo..."

- "Paradigma é um conjunto de regras que estabelecem fronteiras e descrevem como resolver os problemas dentro destas fronteiras."
- "Os paradigmas influenciam nossa percepção; ajudam-nos a organizar e a coordenar a maneira como olhamos para o mundo..."
- "... Programação Orientada a Objetos (POO) é um paradigma para o desenvolvimento de software que baseia-se na utilização de componentes individuais (objetos) que colaboram para construir sistemas mais complexos. A colaboração entre os objetos é feita através do envio de mensagens".

- O paradigma de objetos baseia-se em alguns conceitos, como:
  - Classes
  - Objetos
  - Encapsulamento
  - Herança, dentre outros.

# Classes, Objetos e Métodos

# O que é uma Classe?

"Uma classe é um gabarito para a definição de objetos."



# O que é uma Classe?

"Uma classe é um gabarito para a definição de objetos."

Através da definição de uma classe, descreve-se que **propriedades/atributos** e **métodos/ações** o objeto terá.



# **Definindo uma classe no Ruby**

```
#lobo.rb
class Lobo
...
end
```

### Instanciando uma classe (Objeto)

```
#lobo.rb
class Lobo
end
-----
#app.rb / irb
require_relative 'lobo'
lobo_1 = Lobo.new
lobo 2 = Lobo.new
```

### Definindo uma ação/método em uma classe

```
#lobo.rb
class Lobo
    def uivar
        puts "auuu!"
    end
end
```

### Definindo uma ação/método em uma classe

```
#lobo.rb
class Lobo
   def uivar
      puts "auuu!"
   end
end
#app.rb / irb
require relative 'lobo'
lobo 1 = Lobo.new
lobo 1.uivar
```

# Usando parâmetros uma ação/método

```
#lobo.rb
class Lobo
    def uivar(forca=3) = puts "a#{'u'*forca}!"
end
-----
#app.rb / irb
require_relative 'lobo'
lobo_1 = Lobo.new
lobo 1.uivar(7)
```



### **Encapsulando métodos**

```
#lobo.rb
class Lobo
   def uivar(forca=3)
     puts "a#{'u'*forca}!"
     grunir
   end
   private
   def grunir = puts "arrqqqq!!!"
end
#app.rb / irb
require relative 'lobo'
lobo 1 \equiv Lobo.new
lobo 1.uivar
```

### **Encapsulando métodos**

```
#lobo.rb
class Lobo
   def uivar(forca=3)
     puts "a#{'u'*forca}!"
     grunir
  end
   private
   def grunir = puts "arrgggg!!!"
end
#app.rb / irb
require relative 'lobo'
lobo 1 \equiv Lobo.new
lobo 1.public methods
lobo 1.public methods.include(:uivar)
lobo 1.public methods.include(:grunir)
```



### Herdando classes

```
#animal.rb
class Animal
   def correr = puts "correndo..."
end
#lobo.rb
require relative 'animal'
class Lobo < Animal
end
#app.rb / irb
require relative 'lobo'
lobo 1 \equiv Lobo.new
lobo 1.corre
lobo 1.uiva
```



### **Módulos**

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

Diferente das classes, você não pode criar objetos baseados em módulos nem pode criar módulos que herdam desse módulo.

### **Módulos**

```
#bichos.rb
module Bichos
      MAIS COMUNS = ['cachorro', 'gato']
      class Animal
         def corre = puts "correndo..."
      end
      class Lobo < Animal
      end
end
#app.rb / irb
require relative 'bichos'
include Bichos
Bichos::MAIS_COMUNS
lobo_1 = Lobo.new
lobo_1.corre
lobo_1.uiva
```



### Exercício

Crie um módulo **Matematica** que contenha a classe **Calculadora**, e essa classe deve possuir os métodos **somar**, **subtrair**, **multiplicar** e **dividir** que aceitam dois parâmetros a fim de realizar a operação em questão e mostrar através de um puts. Em seguida, crie uma arquivo app.rb que use esse módulo e classe.