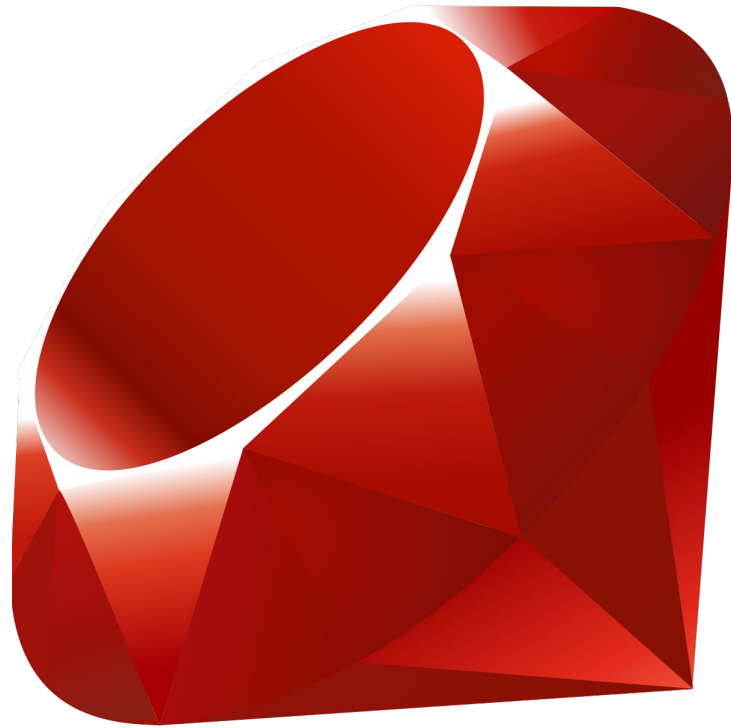


# Ruby

Linguagem



# História

- Desenvolvida em 1995 Yukihiro "Matz" Matsumoto
- Matz queria uma linguagem de script que fosse mais poderosa do que Perl, e mais orientada a objetos do que Python
- Até a v1.9.2-p290 utilizava GNU, apartir da 1.9.3-p0 adotou FreeBSD
- Inicialmente foram propostos dois nomes: "Coral" e "Ruby". Mats Escolheu "Ruby" porque essa era a pedra zodiacal de um de seus colegas.




# Sobre

- Suporta programação funcional, orientada a objetos, imperativa e reflexiva
- Linguagem Interpretada
- Multiparadigma
- Tipagem Forte e Dinâmica
- Gerenciamento de Memória Automática
- Gratuita
- Livre para copiar, modificar e distribuir

# Características

- Todas as variáveis são objetos, onde até os "tipos primitivos" (tais como inteiro, real, entre outros) são classes
- Métodos de geração de código em tempo real, como os "attribute accessors"
- Através do RubyGems, é possível instalar e atualizar bibliotecas com uma linha de comando, de maneira similar ao APT do Debian Linux
- Code blocks (blocos de código) passados como parâmetros para métodos; permite a criação de clausuras
- Mixins, uma forma de emular a herança múltipla
- Tipagem dinâmica, mais forte. Isso significa que todas as variáveis devem ter um tipo (fazer parte de uma classe), mas a classe podem ser alterada dinamicamente



# Aspectos da Orientação a Objetos

# Objetos e Métodos

#criando um objeto

```
objeto = Object.new()
```

#definindo um método

```
def pessoa.vai(lugar)
  puts "indo para " + lugar
end
```

#imprimindo retorno

```
puts pessoa.vai("casa")
```

#refatorando método com interpolação

```
def pessoa.vai(lugar)
  "indo para #{lugar}"
end
```

#definindo um método com múltiplos argumentos

```
def pessoa.troca(roupa, lugar)
  "trocando de #{roupa} no #{lugar}"
end
```

#invocação método múltiplos argumentos

```
pessoa.troca('camiseta', 'banheiro')
```

#método com múltiplos argumentos com valor padrão

```
def pessoa.troca(roupa, lugar='banheiro')
  "trocando de #{roupa} no #{lugar}"
end
```

#invocação com parâmetro

```
pessoa.troca('camiseta', 'banheiro')
```

# invocação sem o parametro:

```
pessoa.troca("camiseta")
```

# Classes e Herança

#criando uma classe

```
class Pessoa
  def fala
    puts "Sei Falar"
  end

  def troca(roupa, lugar="banheiro")
    "trocando de #{roupa} no #{lugar}"
  end
end
```

#instanciando objeto

```
p = Pessoa.new
```

#criando uma classe

```
class Veiculo
```

```
  def acelerar
    puts "Acelerar!"
  end
```

```
end
```

#herdando uma classe

```
class Carro < Veiculo
end
```

# Encapsulamento

#criando uma classe

```
class Pessoa
  def muda_nome(novo_nome)
    @nome = novo_nome
  end

  def diz_nome
    "meu nome é #{@nome}"
  end
end
```

#acessando atributos

```
p = Pessoa.new
p.muda_nome "João"
p.diz_nome
```

#modificadores de atributos

attr\_accessor

attr\_reader

attr\_writer



# Poliformismo

- Não existem interfaces no Ruby
- Utilizam uma abordagem chamada Duck Typing(cada objeto decide por si próprio quais métodos ele possui)

#Exemplo - Implementação mais próxima de uma Interface

```
class MinhaClasse
  def metodo_como_heranca
    raise NotImplementedError, "Implemente esse metodo na classe filho.."
  end
end
```

**IDE's para desenvolvimento Ruby**



**Mão na  
massa!**

<https://github.com/patrickcamargo7/exemplo-crud-rubyonrails>

<http://10.20.15.71:3000>

# Questões

01) Quais os modificadores para tornar atributos privados?

- a) @ e attr\_accessor(reader e write)
- b) private e @
- c) apenas private
- d) protected

# Questões

02) Através de ..... é possível emular herança múltipla na Linguagem Ruby.

- a) attr\_accessor
- b) initialize
- c) mixin
- d) def

# Questões

03) Quando utilizamos o @ na linguagem Ruby, por padrão os atributos criados serão de acesso de tipo:

- a) public
- b) private
- c) protected
- d) outro

# Questões

04) Como é feito a interpolação de String na linguagem Ruby.

- a) `${string}`
- b) `#{string}`
- c) `$string`
- d) `"" + string`

05) Qual o resultado obtido com o código abaixo:

```
class Exemplo
  def imprimir(valor = "Este é o valor definido na passagem de parametro")
    puts ">> #{valor}"
  end
end

exemplo = Exemplo.new
exemplo.imprimir "Este é um valor Qualquer"
```

- a) >> Este é um valor Qualquer
- b) Este é um valor Qualquer
- c) >> Este é o valor definido na passagem de parametro
- d) Este é o valor definido na passagem de parametro



# Referências

<https://www.devmedia.com.br/orientacao-a-objetos-com-ruby/33726>

<https://www.caelum.com.br/apostila-ruby-on-rails/mais-ruby-classes-objetos-e-metodos/#4-2-metodos-comuns>