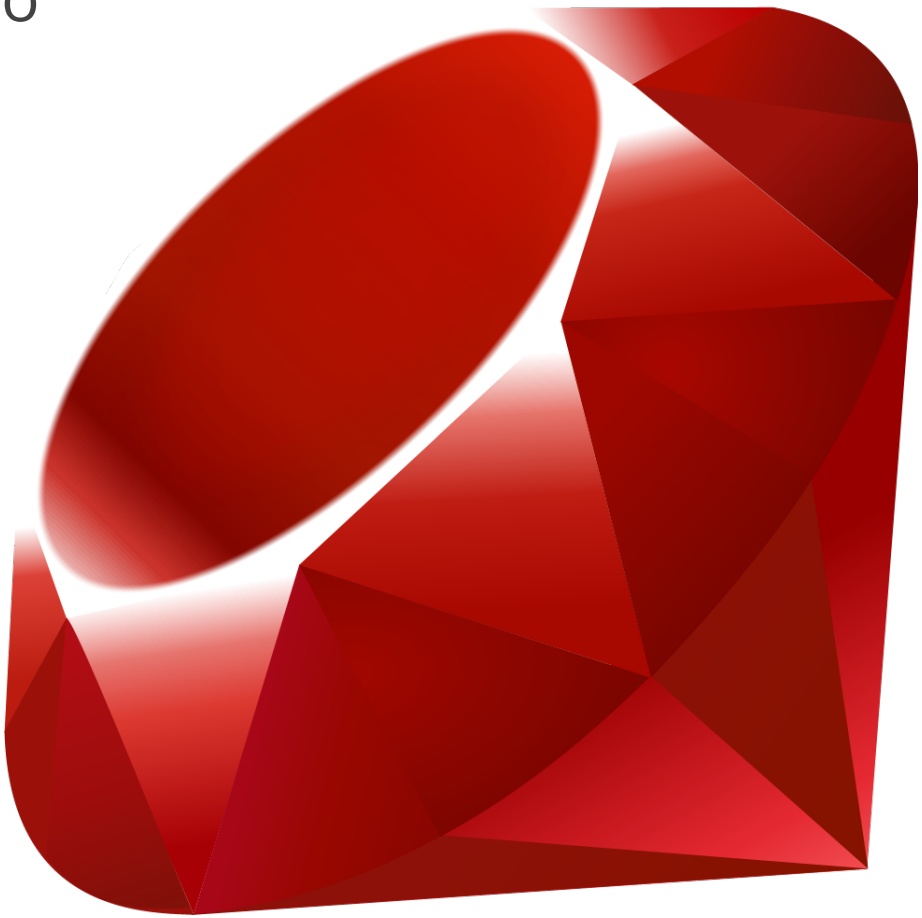
RubyLab #3

- Iteradores e blocos
- Modificadores de acesso
- Syntax Sugar



Breno Martinusso
martinusso.com

Blocos

Uma convenção comum é a utilização de chaves quando um bloco se encaixa em uma única linha, e o uso das palavras-chave **do/end** quando o bloco se estende em linhas.

```
1.upto(10) {|x| puts x }

1.upto(10) do |x|
  puts x
end

1.upto(10)
  {|x| puts x }
```

Blocos e escopo de variáveis

```
total = 0
1.upto(10) {|x| total += x }
puts total
```

Blocos e hashes

```
hash.each do |chave, valor|
  puts "#{chave}: #{valor}"
end
```

Iteradores - números

```
4.upto(6){|x| print x}

3.times {|x| print x }

0.step(3, 0.1) {|x| puts x}
```

Iteradores - textos

```
"breno".each_byte {|c| puts c.chr}

'breno'.split('').each {|c| puts c}

"a\nb\nc\n".each_line{ |l| print l }
```

Criando Iteradores

Para criar funções que possam receber blocos, basta colocar a instrução "**yield**" dentro do corpo da função. Cada vez que o **yield** é executado, ele executa o bloco associado. Para passar parâmetros, eles são colocados logo depois do yield.

```
def calc(a, b)
   yield a, b
end

calc(3, 4) {|x, y| puts x*y }
```

lambda

Mas e se quisermos atribuir um bloco a uma variável?

```
la = lambda{ |param| puts "Olá #{param}" }

la.call("Breno") #=> Olá Breno!
la.class #=> Proc
```

Modificadores de acesso

public

protected

private

```
class Pessoa
  # Métodos públicos aqui

  protected
  # Métodos protegidos aqui

  private
  # Métodos privadas aqui

  public
  # Métodos públicos aqui também
end
```

Métodos acessores e modificadores

Métodos acessores e modificadores são muito comuns e dão a ideia de propriedades.

Existe uma convenção para a definição destes métodos, que a maioria dos desenvolvedores Ruby segue (assim como Java tem a convenção para *getters* e *setters*):

```
class Pessoa
  def nome # acessor
    @nome
  end

  def nome=(novo_nome)
    @nome = novo_nome
  end
end

pessoa = Pessoa.new
pessoa.nome=("José")
puts pessoa.nome
```

Syntax Sugar: facilitando a sintaxe

Priorizando a legibilidade, Ruby abre mão da rigidez sintática **em alguns casos**

```
pessoa.nome=("José")
pessoa.nome= "José"
pessoa.nome = "José"

# Tudo em Ruby é objeto!
10.+(3)
```

42