



1. Introdução ao Ruby

c. Tipos Básicos do Ruby

i. Blocos de Código

1. Um dos recursos mais versáteis e flexíveis o Ruby
2. Iterar em coleções, personalizar o comportamento de métodos e definir “Domain Specific Languages”
3. Para quase tudo são usados blocos
 - a. Até para criar formulários de páginas web no rails e configurar o framework
4. 2 formas:
 - a. { e }
 - b. End
5. Podem receber parâmetros
 - a. Logo na abertura do bloco, entre | |, separados por ,
 - b. Suporte a “Closures reais”
 - i. Variáveis alteradas dentro do bloco serão refletidas para fora
 1. Link para o contexto original
 2. Diferente dos ponteiros de C/C++

6. Exs:

- a.

```
arr = [1, 2, 3, 4]
arr.each { |val|
  print "#{val}\n"
}
1
2
3
4
=> [1, 2, 3, 4]
```

 - i. Bloco simples delimitado por chaves
- b.

```
arr = [1, 2, 3, 4]
arr.each_with_index do |val, idx|
  print "Posição #{idx} valor #{val}\n"
end
Posição 0 valor 1
Posição 1 valor 2
Posição 2 valor 3
Posição 3 valor 4
=> [1, 2, 3, 4]
```

 - i. Praticamente a mesma coisa com outro método de iteração (delimitadores do e end)
- c.

```
valor = 1
arr.each do |val|
```



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

```
valor += val  
end  
=> [1, 2, 3, 4]  
valor  
=> 11
```

- i. Utilizada um closure
- ii. Mantém contexto de onde é chamada
- iii. No próximo exemplo é utilizado um método, nesse caso a variável valor não existe

```
d. valor = 1  
=> 1  
def iterar  
arr = [1, 2, 3, 4]  
arr.each do |val|  
valor += val  
end  
end  
=> nil  
Iterar
```

- i. Também demonstra a nomenclatura de variáveis (nesse caso, local)

7. É possível passar blocos como parâmetros para métodos

- a. Muito usado
- b. Exemplo anterior: passamos um bloco para o each
- c. Criação de um método que recebe um bloco
- d. def recebe_proc_e_passa_parametro

```
if block_given?  
yield  
else  
puts "você precisa passar um bloco para este  
método\n"  
end  
end  
=> nil  
recebe_proc_e_passa_parametro  
você precisa passar um bloco para este método  
=> nil  
recebe_proc_e_passa_parametro { print "dentro do  
bloco" }  
dentro do bloco  
=> nil
```

- e. Passar parâmetros para blocos recebidos nos métodos
- f. def recebe_proc_e_passa_parametro
if block_given?
Yield(23)



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

```
else
puts "você precisa passar um bloco para este
método\n"
end
end
=> nil
recebe_proc_e_passa_parametro do |par|
puts "Recebi #{par} dentro deste bloco\n"
end
Recebi 23 dentro deste bloco
=> nil
```

ii. Procs

1. Parecidos com blocos ou closures simples
2. Diferença: podemos armazenar um proc em uma variável
 - a. Mais "caro" para o Ruby do que simples blocos
 - b. Para o usuário são praticamente iguais
 - c. Não é necessário se preocupar com conversão
3. Ex
 - a.

```
def recebe_proc(&block)
if block
block.call
end
end
=> nil
recebe_proc
=> nil
recebe_proc { print "este bloco vai se tornar uma proc,
pois vai ser atribuído a uma variável no método\n" }
este bloco vai se tornar uma proc, pois vai ser
atribuído a uma variável no método
=> nil
```
4. Criamos procs de duas maneiras
 - a. Construtor

```
p = Proc.new { print "este bloco vai se tornar uma
proc, pois vai ser atribuído a uma variável\n" }
p.call
```
 - b. Palavra-chave lambda (semelhante ao python)

```
p1 = lambda do
print "este bloco vai se tornar uma proc, pois vai ser
atribuído a uma variável\n"
end
p1.call
```
 - c. p.methods
 - d. p.methods.sort



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

iii. Números

1. Todos são objetos (instâncias de alguma classe)

- a. Fixnum
- b. Bignum
- c. Float

2. Ex

- a. `i = 1`
`i.class`
`i1 = 1.1`
`i1.class`
`i2 = 111_222_333`
`i2.class`
`i3 = 999999999999999`
`i3.class`

3. Conversão automática entre Fixnum e Bignum

iv. Valores Booleanos

- 1. true ou false
- 2. nil: nenhum objeto = falso em comparações
- 3. diferente de nil e false é verdadeiro
- 4. ex

- a.

```
def testa_valor(val)
  if val
    print "#{val} é considerado verdadeiro pelo Ruby\n"
  else
    print "#{val} é considerado falso pelo Ruby\n"
  end
end
=> nil
testa_valor true
testa_valor false
testa_valor 1
testa_valor "asda"
testa_valor nil
```

v. Strings

- 1. Representar texto nos códigos Ruby
- 2. 2 tipos: com ou sem expansão de variáveis
- 3. Formas de declarar:
 - a. Aspas “
 - b. Apóstrofes ‘
 - c. <<MARCADOR
 - d. %Q{ }



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

e. `%q{ }`

4. Ex

a. `a = "texto"`

`b = 'texto'`

`c = "texto\nsegunda linha"`

`d = 'texto\nmesma linha'`

`e = "a = #{a} – expansão de variáveis"`

`f = <<__ATE_O_FINAL`

essa é

uma string

bem grande e só termina

quando encontra o marcador `__ATE_O_FINAL`

no início de uma linha

`__ATE_O_FINAL`

`g = %Q{Esta também`

é uma String

com mais de uma linha

e também suporta `#{a}`

expansão de variáveis

}

`h = %q{Já`

esta

que também é multilinha

não suporta `#{a}`

expansão de variáveis}

5. Expansão de variáveis = execução de código ruby dentro de strings

d. Constantes

i. Intervalos numéricos

ii. Arrays

iii. Hashes

iv. Símbolos

v. Expressões Regulares

vi. Classes e métodos

vii. Métodos

e. Módulos

f. Operadores Condicionais e Loops

i. `if / elsif / else / end`

ii. `case / when / else / end`

g. Operadores de Loop

i. `while`



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

- ii. for
 - iii. until
 - iv. begin
 - v. loop
 - vi. Padrões importantes
- h. Dominando o Ruby