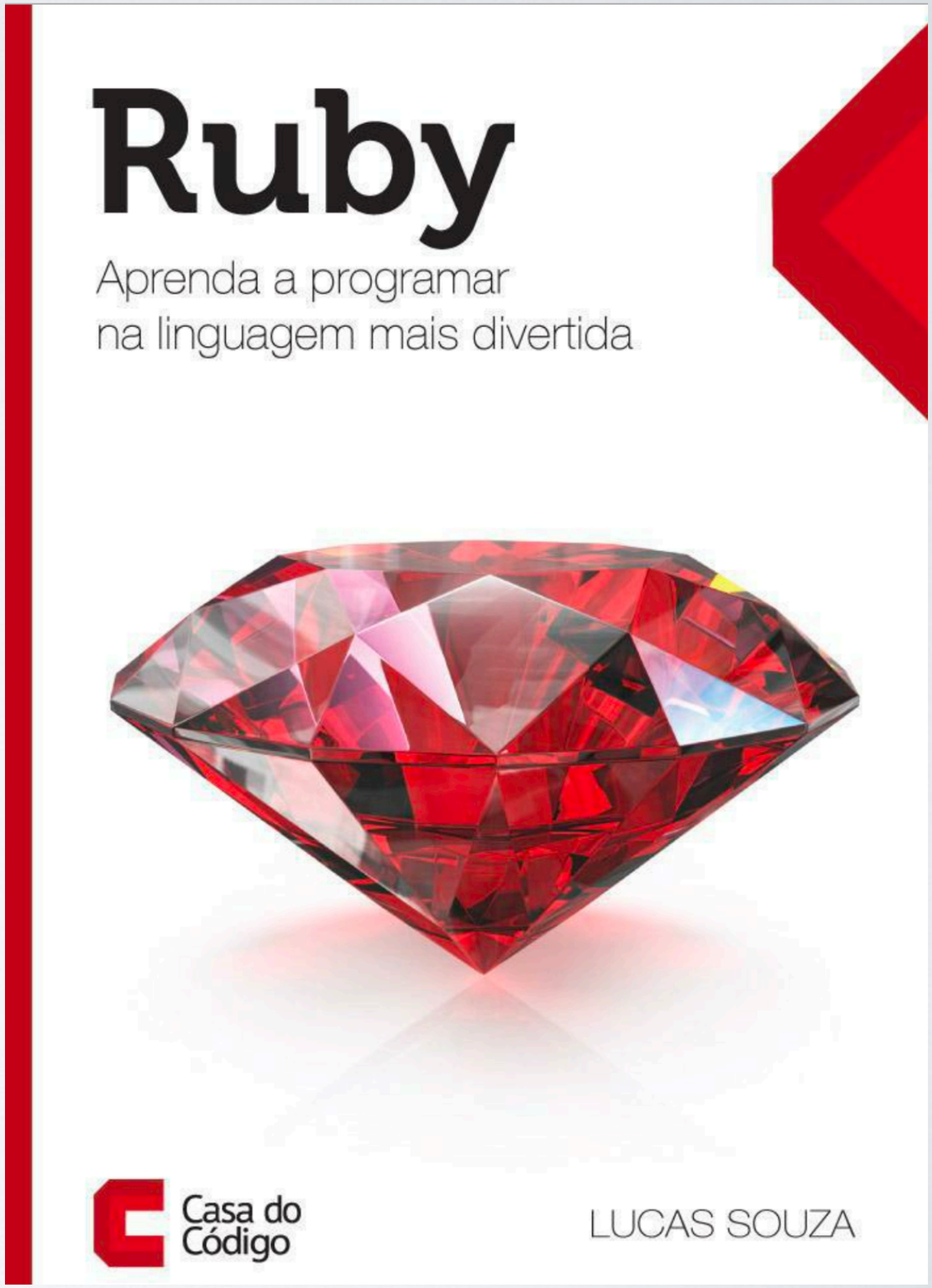
ESTRUTURAS DE DADOS

Rafael Vieira Coelho

(rafaelvc2@gmail.com)



ARRAYS:
CRIANDO UM ARRAY DE STRINGS

Arrays of strings can be created using ruby's percent string syntax:

```ruby
array = %w(one two three four)
```

This is functionally equivalent to defining the array as:

```ruby
array = ['one', 'two', 'three', 'four']
```

`%W` can be used instead of `%w` to incorporate string interpolation. Consider the following:

```ruby
var = 'hello'

%w(#{var}) # => ["\#{var}"]
%W(#{var}) # => ["hello"]
```

Multiple words can be interpreted by escaping the space with a \.

```ruby
%w(Colorado California New\ York) # => ["Colorado", "California", "New York"]
```

ARRAYS:
CRIANDO COM ARRAY::NEW

An empty Array ([ ]) can be created with Array's class method, `Array::new`:

```ruby
Array.new
```

To set the length of the array, pass a numerical argument:

```ruby
Array.new 3 #=> [nil, nil, nil]
```

There are two ways to populate an array with default values:

- Pass an immutable value as second argument.
- Pass a block that gets current index and generates mutable values.

```ruby
Array.new 3, :x #=> [:x, :x, :x]

Array.new(3) { |i| i.to_s } #=> ["0", "1", "2"]

a = Array.new 3, "X"              # Not recommended.
a[1].replace "C"                  # a => ["C", "C", "C"]

b = Array.new(3) { "X" }          # The recommended way.
b[1].replace "C"                  # b => ["X", "C", "X"]
```

ARRAYS: MANIPULANDO ELEMENTOS

Adding elements:

```ruby
[1, 2, 3] << 4
# => [1, 2, 3, 4]

[1, 2, 3].push(4)
# => [1, 2, 3, 4]

[1, 2, 3].unshift(4)
# => [4, 1, 2, 3]

[1, 2, 3] << [4, 5]
# => [1, 2, 3, [4, 5]]
```

ARRAYS:
MANIPULANDO ELEMENTOS

Removing elements:

```ruby
array = [1, 2, 3, 4]
array.pop
# => 4
array
# => [1, 2, 3]
```

```ruby
array = [1, 2, 3, 4]
array.shift
# => 1
array
# => [2, 3, 4]
```

```ruby
array = [1, 2, 3, 4]
array.delete(1)
# => 1
array
# => [2, 3, 4]
```

ARRAYS: MANIPULANDO ELEMENTOS

```ruby
array = [1,2,3,4,5,6]
array.delete_at(2) // delete from index 2
# => 3
array
 # => [1,2,4,5,6]
```

```ruby
array = [1, 2, 2, 2, 3]
array - [2]
# => [1, 3]     # removed all the 2s
array - [2, 3, 4]
# => [1]        # the 4 did nothing
```

ARRAYS: MANIPULANDO ELEMENTOS

Combining arrays:

```
[1, 2, 3] + [4, 5, 6]
# => [1, 2, 3, 4, 5, 6]

[1, 2, 3].concat([4, 5, 6])
# => [1, 2, 3, 4, 5, 6]

[1, 2, 3, 4, 5, 6] - [2, 3]
# => [1, 4, 5, 6]

[1, 2, 3] | [2, 3, 4]
# => [1, 2, 3, 4]

[1, 2, 3] & [3, 4]
# => [3]
```

You can also multiply arrays, e.g.

```
[1, 2, 3] * 2
# => [1, 2, 3, 1, 2, 3]
```

ARRAYS: ACESSANDO ELEMENTOS

You can access the elements of an array by their indices. Array index numbering starts at `0`.

```
%w(a b c)[0] # => 'a'
%w(a b c)[1] # => 'b'
```

You can crop an array using range

```
%w(a b c d)[1..2] # => ['b', 'c'] (indices from 1 to 2, including the 2)
%w(a b c d)[1...2] # => ['b'] (indices from 1 to 2, excluding the 2)
```

This returns a new array, but doesn't affect the original. Ruby also supports the use of negative indices.

```
%w(a b c)[-1] # => 'c'
%w(a b c)[-2] # => 'b'
```

You can combine negative and positive indices as well

```
%w(a b c d e)[1...-1] # => ['b', 'c', 'd']
```

ARRAYS:

CRIANDO UM ARRAY COM LETRAS OU NÚMEROS CONSECUTIVOS

This can be easily accomplished by calling `Enumerable#to_a` on a `Range` object:

```
(1..10).to_a    #=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

`(a..b)` means that it will include all numbers between a and b. To exclude the last number, use `a...b`

```
a_range = 1...5
a_range.to_a        #=> [1, 2, 3, 4]
```

or

```
('a'..'f').to_a    #=> ["a", "b", "c", "d", "e", "f"]
('a'...'f').to_a   #=> ["a", "b", "c", "d", "e"]
```

A convenient shortcut for creating an array is `[*a..b]`

```
[*1..10]           #=> [1,2,3,4,5,6,7,8,9,10]
[*'a'..'f']        #=> ["a", "b", "c", "d", "e", "f"]
```

COMO ADICIONAR NO FINAL (*APPEND*)

```ruby
textos = ['bom dia', 'boa tarde']

textos << 'boa noite'

puts textos
```

```
bom dia
boa tarde
boa noite
[Finished in 0.2s]
```

CRIANDO UMA BIBLIOTECA

```ruby
require_relative 'livro'

biblioteca = []

teste_e_design = Livro.new("Mauricio Aniche", 247, "123454")
web_design_responsivo = Livro.new("Tárcio Zemel", 189, "452565")

biblioteca << teste_e_design
biblioteca << web_design_responsivo

puts biblioteca
```

```
ISBN: 123454
Autor: Mauricio Aniche
Páginas: 247
#<Livro:0x00007fd68e077b98>
ISBN: 452565
Autor: Tárcio Zemel
Páginas: 189
#<Livro:0x00007fd68e077b20>
[Finished in 0.1s]
```

require_relative procura o arquivo livro.rb no diretório corrente.

ENCAPSULANDO A BIBLIOTECA

- attr_acessor permite que não seja preciso criar getters/setters

```ruby
class Biblioteca

    attr_accessor:livros #get/set
    #attr_reader:livros get
    #attr_writer:livros set

    def initialize
        @livros = []
    end

    def adiciona(livro)
        @livros << livro
    end

    def to_s
        for livro in @livros
            puts livro
        end
    end
end
```

ENCAPSULANDO A BIBLIOTECA

```ruby
require_relative 'livro'

biblioteca = []

teste_e_design = Livro.new("Mauricio Aniche", 247, "123454")
web_design_responsivo = Livro.new("Tárcio Zemel", 189, "452565")

biblioteca << teste_e_design
biblioteca << web_design_responsivo

puts biblioteca
```

```ruby
require_relative 'livro'
require_relative 'biblioteca'

biblioteca = Biblioteca.new

teste_e_design = Livro.new("Mauricio Aniche", 247, "123454")
web_design_responsivo = Livro.new "Tárcio Zemel", 189, "452565"

biblioteca.adiciona(teste_e_design)
biblioteca.adiciona web_design_responsivo

puts biblioteca
```

EXERCÍCIO

- Crie um array de alunos (as informações dos alunos devem ser informadas pelo usuário).

- Percorra o array, mostrando o nome e a média de cada aluno (use o método da classe para calcular a média).

ESTRUTURAS DE DADOS

```ruby
require 'set'


numero_sem_repeticao = Set.new [1, 2, 2, 3, 2, 1]

for numero in numero_sem_repeticao do
    p numero
end


# => 1

# => 2

# => 3
```

ADICIONANDO UM DICIONÁRIO (CHAMADO DE HASH EM RUBY)

- Um dicionário é um conjunto de pares no qual se associam chaves a valores.

- Cada chave é única.

- Na biblioteca, teremos um dicionário usando o ISBN como chave.

ADICIONANDO UM DICIONÁRIO (CHAMADO DE HASH EM RUBY)

A Hash can be easily created by using its implicit form:

```
grades = { "Jane Doe" => 10, "Jim Doe" => 6 }
```

Hashes allow an alternate syntax for keys that are symbols. Instead of

```
options = { :font_size => 10, :font_family => "Arial" }
```

You could write it as:

```
options = { font_size: 10, font_family: "Arial" }
```

Each named key is a symbol you can access in hash:

```
options[:font_size]  # => 10
```

A Hash can also be created through its ::new method:

```
grades = Hash.new
grades["Dorothy Doe"] = 9
```

EXERCÍCIO

- Crie um hash que relacione chaves e valores.

- A chave deve ser a matrícula de um aluno e os valores devem ser objetos da classe Aluno (os atributos devem ser 3 notas).

ADICIONANDO UM DICIONÁRIO (CHAMADO DE HASH EM RUBY)

```ruby
class Livro

    attr_accessor :numero_paginas,:categoria

    def initialize(autor, numero_paginas, categoria)
        @autor = autor
        @numero_paginas = numero_paginas
        @categoria = categoria
    end

    def to_s
        puts "ISBN: #{@isbn}"
        puts "Autor: #{@autor}"
        puts "Páginas: #{@numero_paginas}"
    end

    def autor
        @autor
    end

    def autor=(autor)
        @autor = autor
    end
end
```

ADICIONANDO UM DICIONÁRIO (CHAMADO DE HASH EM RUBY)

```ruby
class Biblioteca

    attr_accessor :livros #get/set
    #attr_reader :livros get
    #attr_writer :livros set

    def initialize
        @livros = {}
    end

    def adiciona(livro, isbn)
        @livros[isbn] = livro
    end

    def to_s
        for isbn in @livros.keys
            puts '='*40
            puts "ISBN: #{isbn}"
            puts @livros[isbn]
            puts '='*40
        end
    end
end
```

@livros.values retorna os valores, @livros.keys retorna as chaves

ADICIONANDO UM DICIONÁRIO (CHAMADO DE HASH EM RUBY)

```ruby
require_relative 'livro'
require_relative 'biblioteca'

biblioteca = Biblioteca.new

teste_e_design = Livro.new("Mauricio Aniche", "123454", "Suspense")
web_design_responsivo = Livro.new "Tárcio Zemel", "452565", "Biografia"

biblioteca.adiciona(teste_e_design, 247)
biblioteca.adiciona web_design_responsivo, 189

puts biblioteca
```

```
========================================
ISBN: 247
Autor: Mauricio Aniche
Páginas: 123454
Categoria: Suspense
#<Livro:0x00007fa052909ff8>
========================================
========================================
ISBN: 189
Autor: Tárcio Zemel
Páginas: 452565
Categoria: Biografia
#<Livro:0x00007fa052909f58>
========================================
#<Biblioteca:0x00007fa05290a0c0>
[Finished in 0.1s]
```

EXERCÍCIO

- Alterem o código main3.rb fazendo com que o usuário possa informar dois livros e adicionar os mesmos a biblioteca.

COMO COMPARAR OBJETOS

==, equal? ou eql?

O método == retorna `true` apenas se os dois objetos envolvidos na comparação forem a mesma instância, este é seu comportamento padrão. Ele pode ser sobrescrito afim de efetuar a comparação de outras maneiras.

O método `equal?` é similar ao ==, ele retorna `true` apenas se os dois objetos envolvidos na comparação forem a mesma instância. As bibliotecas existentes na linguagem, quando tem a necessidade de comparar se dois objetos são a mesma instância usam o método `equal?`, por esse motivo, não devemos sobrescrever este método, pois o efeito pode ser bastante prejudicial.

Por fim o método `eql?` por padrão compara as instâncias dos objetos também. Porém, este método **deve** ser sobrescrito quando desejamos avaliar se dois objetos são iguais por seus valores, como fizemos com a classe `Livro`.

EXEMPLO

```ruby
class Livro

    attr_accessor :numero_paginas,:categoria

    def initialize(autor, numero_paginas, categoria)
        @autor = autor
        @numero_paginas = numero_paginas
        @categoria = categoria
    end

    def to_s
        puts "Autor: #{@autor}"
        puts "Páginas: #{@numero_paginas}"
        puts "Categoria: #{@categoria}"
    end

    def autor
        @autor
    end

    def autor=(autor)
        @autor = autor
    end

    def ==(other)
        self.class === other and
        other.autor == @autor and
        other.categoria == @categoria
    end

    alias eql? ==

    def hash
        @autor.hash ^ @categoria.hash # XOR
    end
end
```

EXEMPLO DE COMPARAÇÃO

```ruby
require_relative 'livro'

livro1 = Livro.new("Mauricio Aniche", "123454", "Suspense")
livro2 = Livro.new "Tárcio Zemel", "452565", "Biografia"
livro3 = Livro.new("Mauricio Aniche", "123454", "Suspense")

if livro1 == livro2
    puts 'IGUAIS'
else
    puts 'DIFERENTES'
end

if livro1 == livro3
    puts 'IGUAIS'
else
    puts 'DIFERENTES'
end
```

```
DIFERENTES
IGUAIS
[Finished in 0.1s]
```

COMPARAÇÃO DE OBJETOS

Podemos também usar o módulo Comparable!

```ruby
class Rectangle
  include Comparable

  def initialize(a, b)
    @a = a
    @b = b
  end

  def area
    @a * @b
  end

  def <=>(other)
    area <=> other.area
  end
end

r1 = Rectangle.new(1, 1)
r2 = Rectangle.new(2, 2)
r3 = Rectangle.new(3, 3)

r2 >= r1 # => true
r2.between? r1, r3 # => true
r3.between? r1, r2 # => false
```

EXERCÍCIO

- Vamos criar os métodos que permitam comparar objetos bibliotecas alterando a classe Biblioteca.

- Vamos testar o método criado anteriormente criando 3 bibliotecas, sendo duas iguais.