# **Ruby Avançado II**

#### Assunto da aula

Nesta segunda parte da aula de conteúdos avançados na linguagem Ruby, você aprenderá a utilizar expressões regulares, realizar funções matemáticas, trabalhar com data e hora, tratar chamadas de métodos que não existem, o que é a variável self e outros tipos de métodos ainda não vistos.

#### Conteúdo

* Regex
* Math
* Time
* Method Missing
* Self
* Private and Protected

#### Regex

Abreviatura de **Regular Expressions**(Expressões Regulares), regex é uma sequência de caracteres especiais que nos ajudam a identificar (e selecionar) padrões de caracteres em strings.

#### 3 Maneiras de criar um Regex

1 – Utilizando o formato **/…/**

**ex:**

|  |  |
| --- | --- |
| 1 | /expressao/ |

Geralmente a mais utilizada.

**2 –**Através do formato**%r{…}**

**Ex:**

|  |  |
| --- | --- |
| 1 | %r{expressao} |

3 – Utilizando um Construtor através do Regexp.new(‘…’)

**Ex:**

|  |  |
| --- | --- |
| 1 | Regexp.new('expressao') |

#### Casamento de Padrões

É um ato que verifica a presença de padrões em uma cadeia de caracteres.

Pode ser realizado através do operador **=~** ou do método **match**

### Operador =~

1- O operador **=~** retorna o offset, ou seja, a distância entre o começo da string até o local onde ocorre o casamento de padrão especificado na expressão.

|  |  |
| --- | --- |
| 1 | /by/ =~ 'ruby' |

2- A mesma comparação também pode ser realizada iniciando pela string.

|  |  |
| --- | --- |
| 1 | 'ruby' =~ /by/ |

3- Quando não ocorre o casamento de padrão, o resultado é **nulo**

|  |  |
| --- | --- |
| 1 | 'ruby' =~ /rails/ |

### Método match

1- Retorna um objeto do tipo **MatchData,**contendo todos os resultados do casamento de padrão.

|  |  |
| --- | --- |
| 1  2  3 | phrase = "Hellow, how are you?"    match\_data = /how/.match(phrase) |

2- Em um objeto do tipo MatchData você pode, por exemplo, acessar o que vem antes do casamento através do método **pre\_match**

|  |  |
| --- | --- |
| 1 | match\_data.pre\_match |

3- Ou acessar o que vem depois do casamento com **post\_match**

|  |  |
| --- | --- |
| 1 | match\_data.post\_match |

### Metacharacters and Escapes

Os símbolos **(, ), [, ], {, }, ., ?, +, \*,**são **metacharacters.**Eles possuem um significado quando utilizados em expressões regulares.

1- Caso o padrão que você procura seja um **metacharacter**, utilize o símbolo de **escap**e **\**para realizar a busca

|  |  |
| --- | --- |
| 1  2  3 | /\?/.match('Tudo bem?')    /bem\!\!\!/.match('Muito bem!!!') |

### Character Classes

É uma lista que informa quais caracteres devem aparecer em um ponto do casamento.

1- Uma **character class** é delimitada por colchetes [, ]

|  |  |
| --- | --- |
| 1 | /[t]exto/.match('texto começando com t') |

2- Você pode especificar um **range** com o símbolo **–**

|  |  |
| --- | --- |
| 1 | /[1-5]/.match('123') |

Assim, é procurado um padrão onde o primeiro número pode ser 1, 2, 3, 4 ou 5

3- Note que o casamento também ocorre em uma string que começa com o número 2

|  |  |
| --- | --- |
| 1 | /[1-5]/.match('223') |

4- O range pode ser utilizado para letras

|  |  |
| --- | --- |
| 1 | /[a-z]/.match('Oi') |

Existem alguns **metacharacters** que se comportam como **character classes**

5- Por exemplo, o **metacharacter** **\d** verifica o padrão [0-9]

|  |  |
| --- | --- |
| 1 | /A\d/.match('A4') |

### Repetition

É possível definir a repetição de um mesmo padrão, evitando escrever a mesma coisa diversas vezes.

1- Por exemplo, verifique o casamento de padrão em uma etiqueta que começa com uma letra e 3 números.

|  |  |
| --- | --- |
| 1 | "A343".match(/[A-Z]\d{3}/) |

Ao em vez de digitar \d\d\d você pode escrever**\d{3}**

2- Verifique se uma letra foi escrita mantendo o padrão de repetição entre três ou mais vezes

|  |  |
| --- | --- |
| 1 | "BBB AAAA".match(/A{3,}/) |

A vírgula em **{3, }** informa que a repetição pode ocorrer 3 ou mais vezes

Conclui-se que a Expressão Regular é uma ferramenta muito poderosa, capaz de identificar qualquer padrão em uma string. Caso se interesse pelo tema, recomendo fortemente que leia esta [documentação](https://ruby-doc.org/core-2.5.1/doc/regexp_rdoc.html).

#### Math

Math é um módulo nativo para funções matemáticas.

1- Sabendo a raiz quadrada de 64

|  |  |
| --- | --- |
| 1 | Math.sqrt(64) |

2- Verificando o logaritmo de 10000 na base 10

|  |  |
| --- | --- |
| 1 | Math.log10(10000) |

3- O logaritmo de 16 na base 2

|  |  |
| --- | --- |
| 1 | Math.log2(16) |

4- Calculando o cosseno para o ângulo de 30º

Primeiro transforme o ângulo em um valor radiano

|  |  |
| --- | --- |
| 1 | radian = 30 \* (Math::PI / 180) |

Depois utilize o método **cos**

|  |  |
| --- | --- |
| 1 | Math.cos(radian) |

O módulo Math também fornece o valor de duas constantes bastante utilizadas:

**E** e **PI**

5- Para consultar o valor da constante **E**execute

|  |  |
| --- | --- |
| 1 | Math::E |

2- Consulte o valor da constante **PI** com a instrução

|  |  |
| --- | --- |
| 1 | Math::PI |

Tenha acesso a mais métodos matemáticos do módulo Math através da [documentação](https://ruby-doc.org/core-2.2.0/Math.html).

#### Time

O Ruby conta com uma classe chamada **Time**para representar datas e horas.

A seguir você verá como é natural utilizar alguns métodos dessa classe para obter informações do tempo.

### Trabalhando com times

1- Para imprimir o horário atual execute

|  |  |
| --- | --- |
| 1  2 | time = Time.now  puts time |

### 2- Para o ano use o método year

|  |  |
| --- | --- |
| 1 | puts time.year |

3- Imprimindo o mês do ano com o método **month**

|  |  |
| --- | --- |
| 1 | puts time.month |

4- Para saber o dia do mês utilize o método **day**

|  |  |
| --- | --- |
| 1 | puts time.day |

5- O método **strftime** permite a formatação de uma data em uma forma específica. Essa formatação é feita por diretivas que começam com o símbolo de %.

|  |  |
| --- | --- |
| 1 | time.strftime('%d/%m/%y') |

Significado das diretivas

%d -> Dia do mês

%m -> Mês do ano

%y -> Ano

6- Verificando se o dia da semana é sábado

|  |  |
| --- | --- |
| 1 | puts time.saturday? |

Você também pode comparar duas datas.

7- Crie um novo objeto com o horário atual

|  |  |
| --- | --- |
| 1 | time2 = Time.now |

8- Ao comparar as datas de forma completa o resultado será **false**

|  |  |
| --- | --- |
| 1 | time == time2 # => false |

9- Comparando o ano das duas datas, o resultado será **true**

|  |  |
| --- | --- |
| 1 | time.year == time2.year # => true |

#### Missing Methods

Voltando para o conteúdo envolvendo classes e métodos, conheça agora o método **method\_missing.**Ele é utilizado para interceptar chamadas a métodos que não existem.

Para saber**como implementar este método:**

1- Crie um arquivo chamado **method\_missing.rb** com o seguinte código

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class Fish   def method\_missing(method\_name)     puts "Fish don't have #{method\_name} behavior"   end     def swim     puts 'Fish is swimming'   end  end    fish = Fish.new  fish.swim  fish.walk |

A partir daí você pode controlar como responder este tipo de situação.Quando o método **walk** que não existe é chamado, o método **method\_missing**faz uma interceptação. Nele é recebido como parâmetro o nome do método que não existe.

#### Self

No ruby, self é uma **variável especial** que **aponta** para o **objeto atual**.

1- Crie o arquivo **self.rb** com o seguinte código

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class Foo   attr\_accessor :teste   def bar     puts self   end  end    foo = Foo.new  puts foo  foo.bar |

### A variável self aponta para o Objeto onde ela se encontra.  No caso está apontando para uma instância da classe Foo e, por isso, as instruções puts foo e puts self retornam o mesmo resultado.

Perceba que a variável self **não precisa** ser declarada. Ela é disponível em qualquer lugar, mas não esqueça que seu valor é referente ao objeto que pertence.

**Variáveis self em objetos diferentes possuem valores diferentes.**

2- Com o **self** é possível criar **métodos de classe**,  que não precisam de uma instância para serem chamados.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class Foo   def self.bar     puts self   end  end    Foo.bar |

**3- Você também pode utilizá-lo para se referir a atributos do objeto atual.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class Pen   attr\_accessor :color   def pen\_color     puts "The color is " + self.color   end  end    pen = Pen.new  pen.color = "blue"  pen.pen\_color |

O **self.color** retorna a cor do objeto **pen.**

#### Métodos Private e Protected

Por padrão, todos os métodos definidos são públicos. Isso significa que eles podem ser acessados por qualquer um.

Mas além dos métodos públicos, existem outros dois tipos de métodos chamados **private** e **protected.**

### Private

Método interno de uma classe. Apenas os métodos públicos dessa classe ou de classes descendentes podem chamá-lo.

O **self**é o único receptor de um método private

1- Defina um método abaixo da palavra **private**, para que ele seja deste tipo

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class Foo   def call\_private     bar   end     private     def bar     puts "private method"   end  end    foo = Foo.new    foo.call\_private |

Se você tentar chamá-lo a partir da instância foo (**foo.bar**), receberá uma mensagem de erro informando que este método é privado.

### Protected

A **diferença** entre ele e o **private**, é que o método **protected**pode ter como receptor qualquer instância de sua classe.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | class Foo   def call\_private(instance)     instance.bar   end     protected     def bar     puts "private method"   end  end    instance\_1 = Foo.new    instance\_2 = Foo.new    instance\_1.call\_private(instance\_1)    instance\_1.call\_private(instance\_2) |

Perceba que você pode utilizar o receptor **instance** para o método **bar**que é do tipo **protected,**por isso,tome cuidado ao utilizar um método **protected**.

#### Missões especiais

### Missão 1

Crie uma expressão regular que faça o casamento de padrão com o número de telefone presente no texto a seguir.

**“Olá, tudo bem? Meu whats app é (99) 7 4321-1234”**

### Missão 2

Em uma classe chamada **Carro**, crie um método público chamado **get\_km**que recebe como parâmetro a seguinte informação “Um fusca de cor amarela viaja a 80km/h ”.

O método **get\_km**deve chamar um **método privado** com o nome de **find\_km**. Este deve localizar e retornar o casamento de padrão **80km/h**.

No final, imprima este retorno.