

# Processamento de Linguagem Natural

## Expressões Regulares

Yuri Malheiros (yuri@ci.ufpb.br)

# Introdução

- Expressão Regular (regex) é uma linguagem para especificar padrões textuais que podem ser buscados em texto
  - Disponível nas principais linguagens de programação
  - Disponível em várias ferramentas
- Usando regex podemos:
  - Pesquisar em um corpus e retornar uma ou todas as ocorrências encontradas
  - Verificar se um string satisfaz um determinado padrão

# Expressões Regulares

- O tipo mais simples de expressão regular é uma sequência de caracteres
- Por exemplo, para pesquisar pela palavra "cidade" basta usar a expressão regular **cidade**
- Vamos utilizar a ferramenta RegExr para testar nossas expressões
  - <https://regexr.com>

# Expressões Regulares

Expression

<> JavaScript ▾

🚩 Flags ▾

/cidade/g

Text

Tests

1 match (0.0ms)

Moro • na • cidade • de • João • Pessoa

# Expressões Regulares

- Utilizando apenas a letra **a**, temos:

Expression

<> JavaScript ▾

Flags ▾

/a/g

Text

Tests

3 matches (0.0ms)

Moro•na•cidade•de•João•Pessoa

# Expressões Regulares

- Expressões regulares são case sensitive
  - A expressão **cidade** é diferente de **Cidade**
- Usando a expressão **cidade** não serão encontradas ocorrências de "Cidade" (com C maiúsculo)

# Expressões Regulares

- Podemos resolver isso utilizando disjunção de caracteres
- Nas expressões regulares, caracteres colocados entre colchetes definem uma disjunção
  - O padrão **[cC]** casa com o "c" minúsculo ou o "C" maiúsculo
- Com isso, o padrão **[cC]idade** vai encontrar ocorrências tanto da palavra "cidade" quanto de "Cidade"

Expression

<> JavaScript ▾

Flags ▾

/[cC]idade/g

Text

Tests

2 matches (0.0ms)

Moro•na•cidade•de•João•Pessoa•.•Cidade•do•litoral•nordestino•|

Expression

<> JavaScript ▾

Flags ▾

/[abcd]/g

Text

Tests

13 matches (1.0ms)

Moro•na•cidade•de•João•Pessoa•.•Cidade•do•litoral•nordestino•.

Expression

<> JavaScript ▾

Flags ▾

/[0123456789]/g

Text

Tests

10 matches (0.0ms)

0•telefone•dele•é•83•1234•5678



# Expressões Regulares

- A expressão regular **[0123456789]** especifica qualquer dígito
- Para letras temos **[abcdefghijklmnopqrstuvwxyz]**
- Muito grande para ser digitado

# Expressões Regulares

- Como padrões para sequência de dígitos ou letras são muito comuns existe uma forma simplificada de especificá-los
- **[2-5]** define qualquer dígito de 2 até 5 (incluindo eles)
- **[b-g]** define qualquer letra de b até g (incluindo elas)

# Expressões Regulares

- Assim, podemos definir também:
  - **[a-z]** - casa com qualquer letra minúscula
  - **[A-Z]** - casa com qualquer letra maiúscula
  - **[0-9]** - casa com qualquer dígito

# Expressões Regulares

- Como fazer para casar com qualquer letra não importando se é maiúscula ou minúscula?
- **[a-zA-Z]** - casa com letras de a até z ou de A até Z

**Expression**<> JavaScript ▾🚩 Flags ▾

`/[a-zA-Z]/g`

**Text****Tests**

13 matches (0.0ms)

0 • telefone • dele • é • 83 • 1234 • 5678

# Expressões Regulares

- Utilizando colchetes também podemos especificar exclusão de caracteres
- A expressão **[^a]** casa com qualquer caractere com exceção do “a”
  - Notem o ^ logo após a abertura dos colchetes

Expression

<> JavaScript ▾

Flags ▾

[^a-z]

/g

Text

Tests

6 matches (0.0ms)

Processamento•de•Linguagem•Natural

Expression

<> JavaScript ▾

Flags ▾

[^nN]

/g

Text

Tests

31 matches (1.0ms)

Processamento•de•Linguagem•Natural

Expression

<> JavaScript ▾

Flags ▾

[a^]

/g

Text

Tests

6 matches (0.0ms)

Olhe•para•cima•^•agora

# Expressões Regulares

- Usamos a disjunção para encontrar palavras não importando se elas começam com letra maiúscula ou minúscula
- Como podemos tratar palavras no plural?
- A expressão **carro** não serve para encontrar a palavra “carros” (no plural)

# Expressões Regulares

- O quantificador **?** nos permite definir um caractere como opcional
  - A expressão regular **carros?** casa com "carro" e "carros"
- A interrogação especifica que o caractere anterior pode aparecer ou não
  - A expressão **carr?o** casa com "carro" e "caro"



# Expressões Regulares

**Expression** JavaScript Flags

`/carros?/g`

Text Tests

2 matches (0.0ms)

carro•e•carros

**Expression** JavaScript Flags

`/carr?o/g`

Text Tests

2 matches (0.0ms)

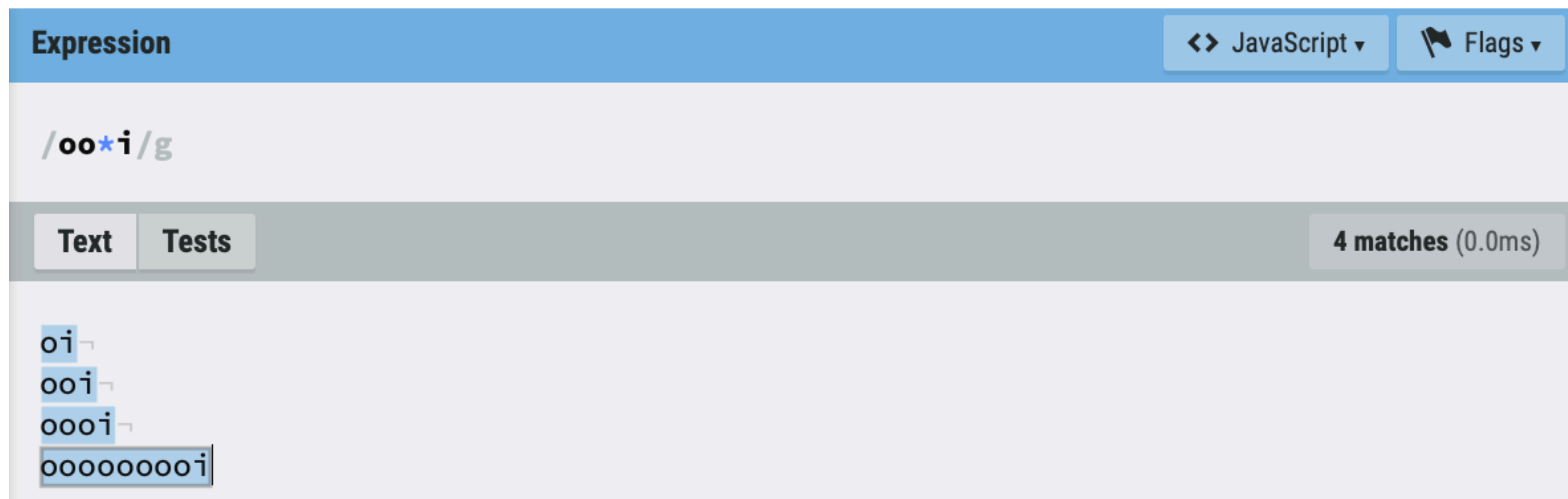
carro•e•caro

# Expressões Regulares

- Considere que queremos encontrar num texto informal todas as saudações "oi"
  - Mas sabemos que algumas pessoas podem usar variações como:  
"ooi", "ooooi", "oo ... ooi"
- Nesse caso, temos que a letra "o" aparece no início, a letra "i" no fim, e entre elas podemos ter zero ou mais letras "o"

# Expressões Regulares

- O quantificador **\*** especifica que o caractere anterior pode aparecer zero ou mais vezes
- Assim, a expressão regular **oo\*i** casa com “oi”, “ooi”, “oooi”, “ooooi” ...



# Expressões Regulares

- Como criar uma expressão regular para buscar números?
  - Eles podem ter apenas um dígito, por exemplo, 5
  - Ou podem ter vários, por exemplo, 72392
- Dica: podemos utilizar colchetes e \* em um mesmo padrão

# Expressões Regulares

- Para um dígito sabemos que **[0-9]** é a solução
- Para mais dígitos podemos usar: **[0-9][0-9]\***
- Por que não usar apenas **[0-9]\*** ?

# Expressões Regulares

- Muitas vezes é inconveniente ter que digitar um padrão duas vezes para garantir que ele apareça uma vez ou mais
- Para isso, existe o quantificador **+**
- Ele especifica que o caractere anterior pode aparecer uma ou mais vezes
- Assim, o padrão para detectar um número se resume a **[0-9]+**

# Expressões Regulares

- Um caractere especial importante nas expressões regulares é o ponto: .
- Ele casa com qualquer caractere
- Por exemplo, o padrão **alun.s** casa com "alunos", "alunas", "alunxs", "alun#s", etc.
- Como fazer para funcionar apenas com "alunos" e "alunas" ?

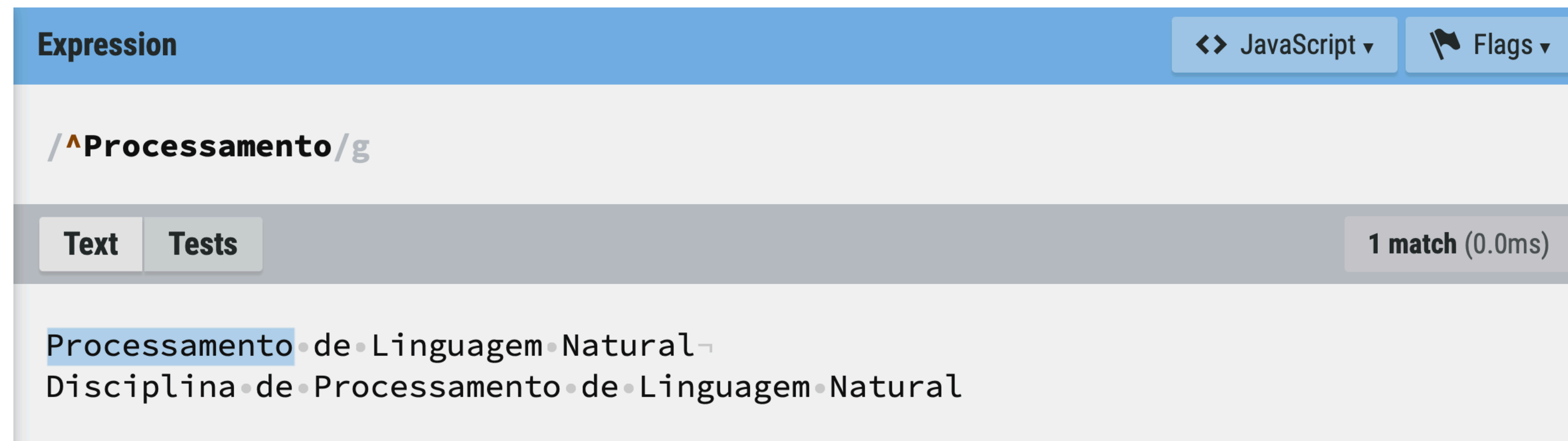
# Expressões Regulares

- As expressões regulares vão procurar caracteres em qualquer parte de um texto
- Existem caracteres especiais chamados de âncoras que definem lugares específicos de um texto



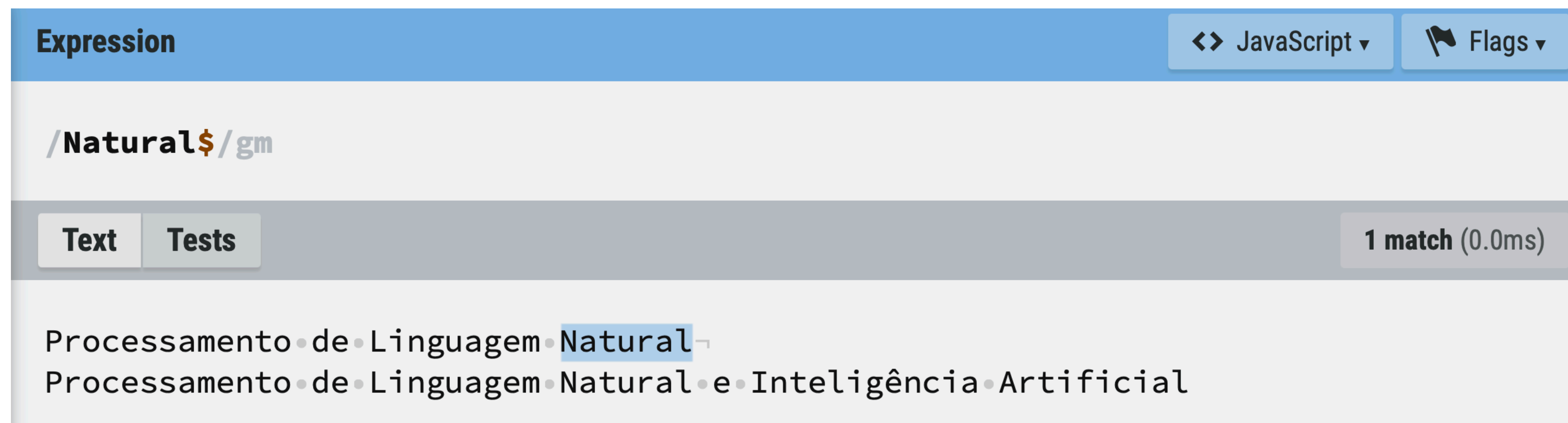
# Expressões Regulares

- O caractere **^** especifica o início da linha
- A expressão **^Processamento** casa apenas com a palavra Processamento se ela estiver no início de uma linha



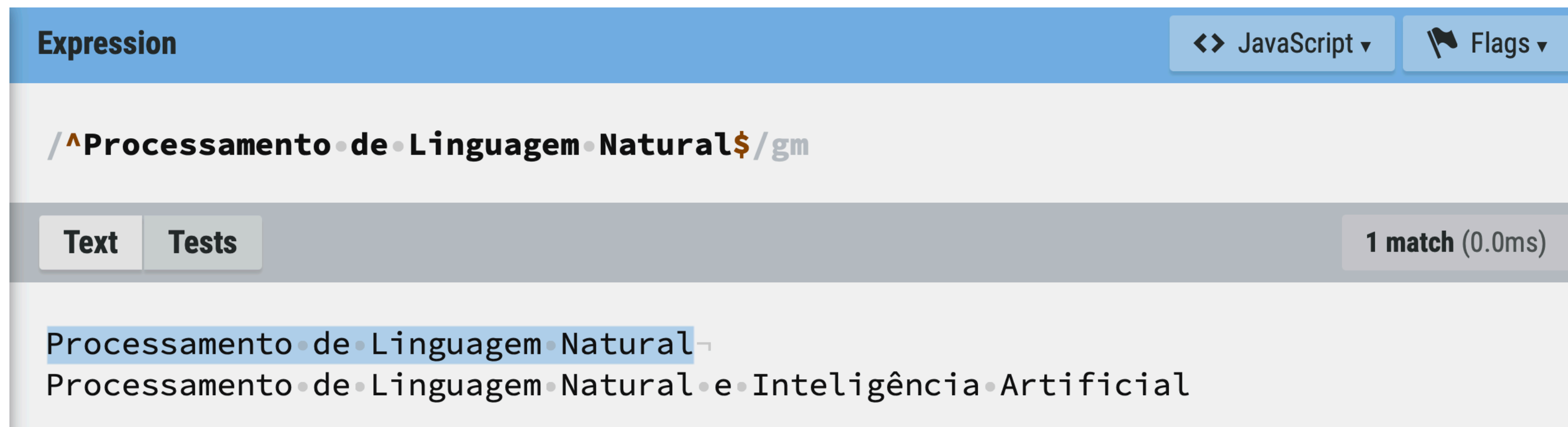
# Expressões Regulares

- O caractere **\$** especifica o fim da linha
- A expressão **Natural\$** casa apenas com a palavra trabalho se ela estiver no fim de uma linha



# Expressões Regulares

- Combinando os dois podemos especificar um padrão para uma linha completa:
- A expressão **`^Processamento de Linguagem Natural$`** casa com uma linha que contém apenas o texto “Processamento de Linguagem Natural”

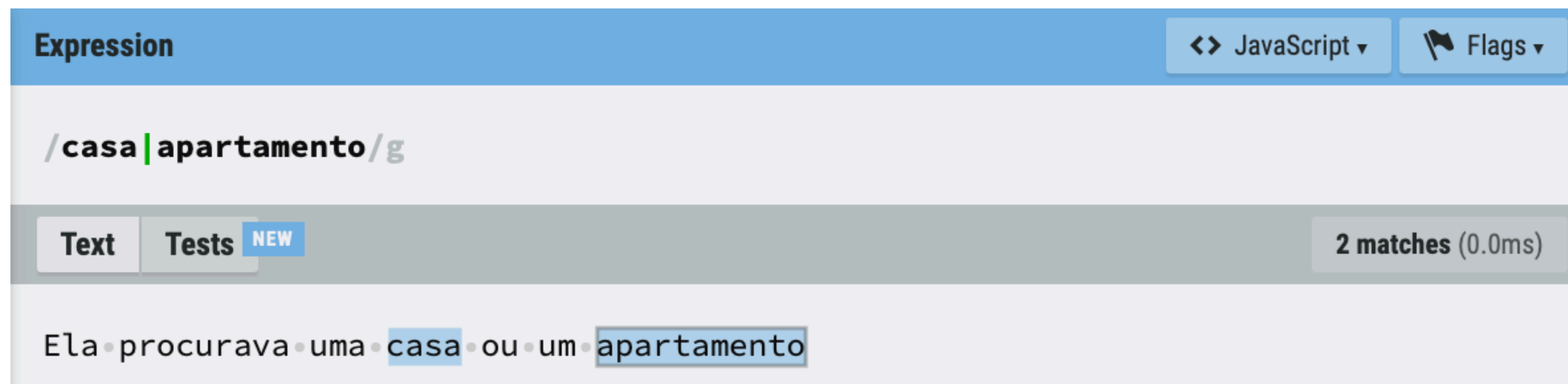


# Expressões Regulares

- Suponha que precisamos procurar em textos sobre imóveis a ocorrência das palavras “casa” e “apartamento”
- Neste caso também temos uma disjunção (casa ou apartamento), mas os colchetes trabalham no nível de caractere, não de palavras

# Expressões Regulares

- Para isso usamos o operador de disjunção |
- O padrão **`casa|apartamento`** encontra tanto “casa” quanto “apartamento”

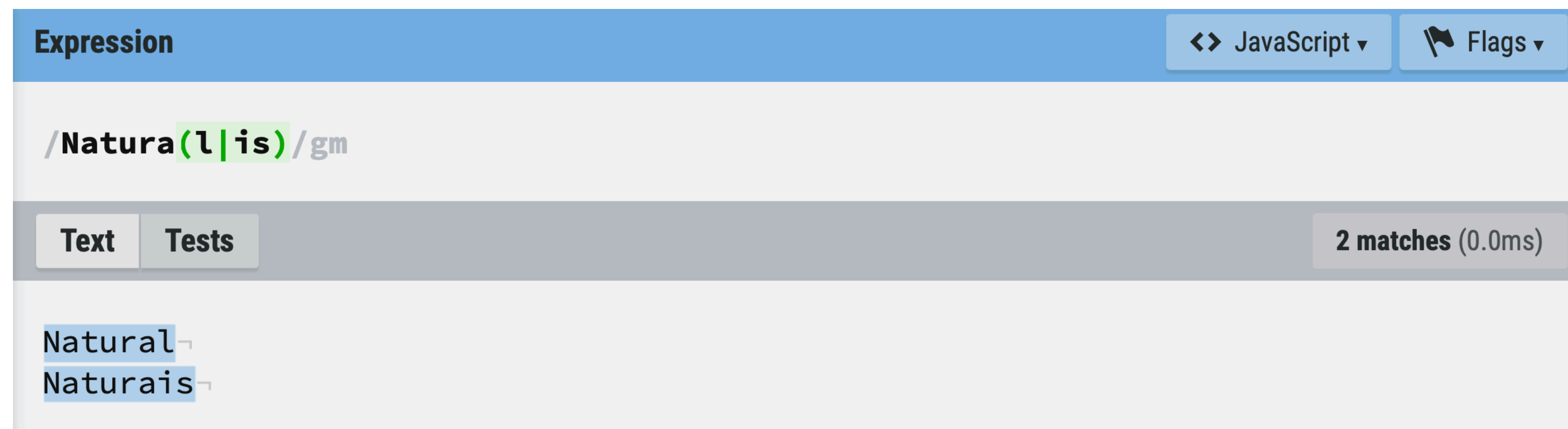


# Expressões Regulares

- As vezes precisamos usar o operador de disjunção no meio de uma sequência maior
- Como fazer uma expressão para casar com “natural” e “naturais”?

# Expressões Regulares

- Podemos simplesmente usar **natural|naturais**
- Mas podemos ser mais concisos utilizando parênteses
- **natura(l|is)** - esse padrão casa com os caracteres “natura” seguido de “l” ou “is”



# Expressões Regulares

- Colocar um padrão entre parênteses faz com que ele se comporte como um único caractere
- Isto é especialmente útil para os quantificadores: **?**, **\*** e **+**



# Exercício

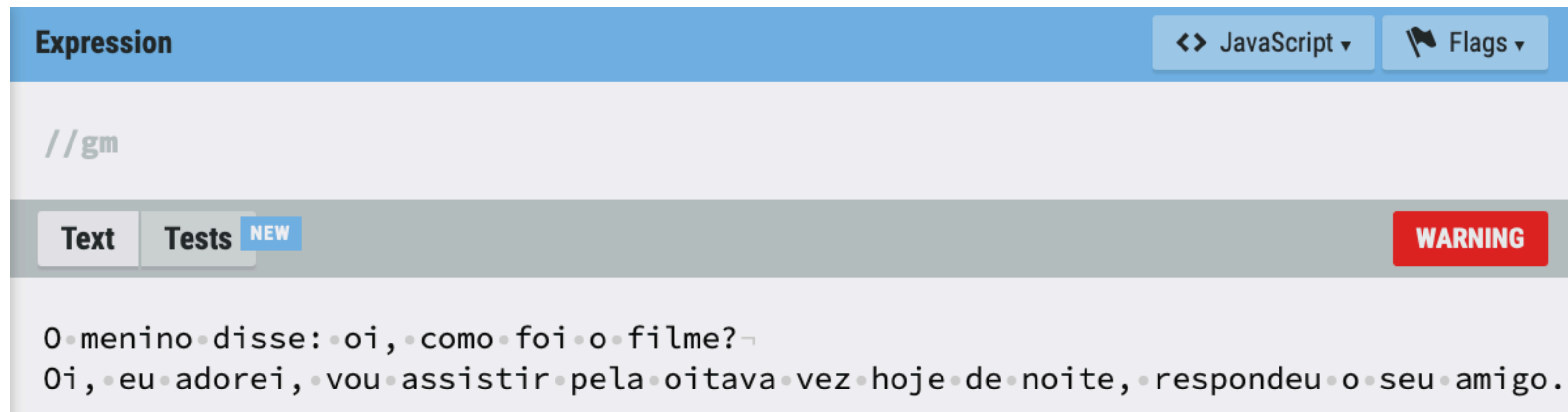
- Como encontrar um string que traz uma lista de nomes de pessoas da seguinte forma:
  - ana;pedro;maria;lucas;

# Exercício

- Como encontrar um string que traz uma lista de nomes de pessoas da seguinte forma:
  - ana;pedro;maria;lucas;
- **([a-z]+;)+**
  - **[a-z]+;** - uma sequência de um ou mais caracteres minúsculos terminando com um ;
  - **([a-z]+;)+** - o padrão anterior pode aparecer uma ou mais vezes

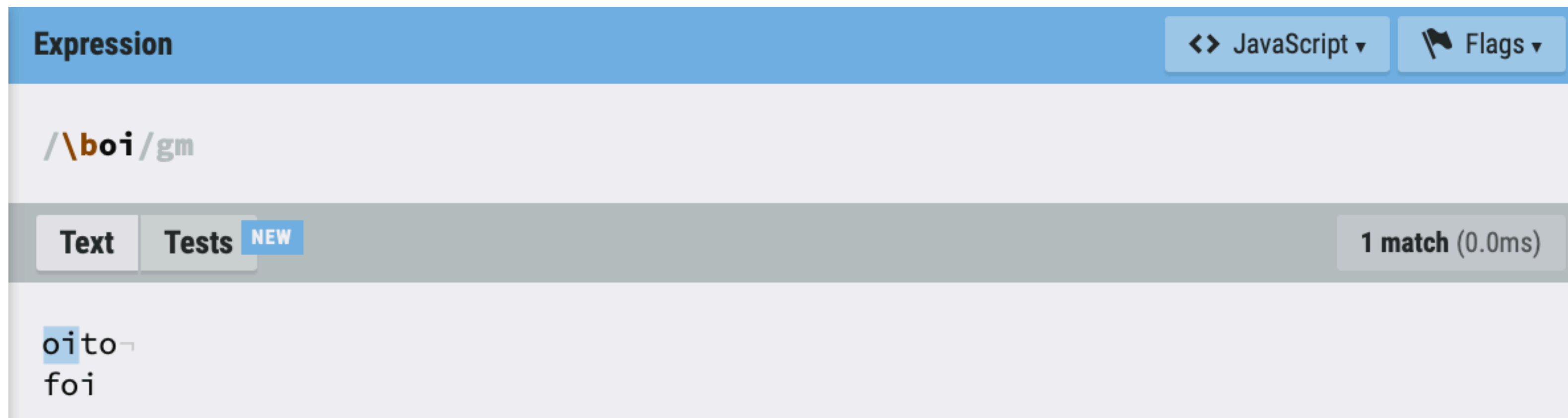
# Expressões Regulares

- Como criar um padrão que encontre todas as palavras “oi” no texto abaixo:



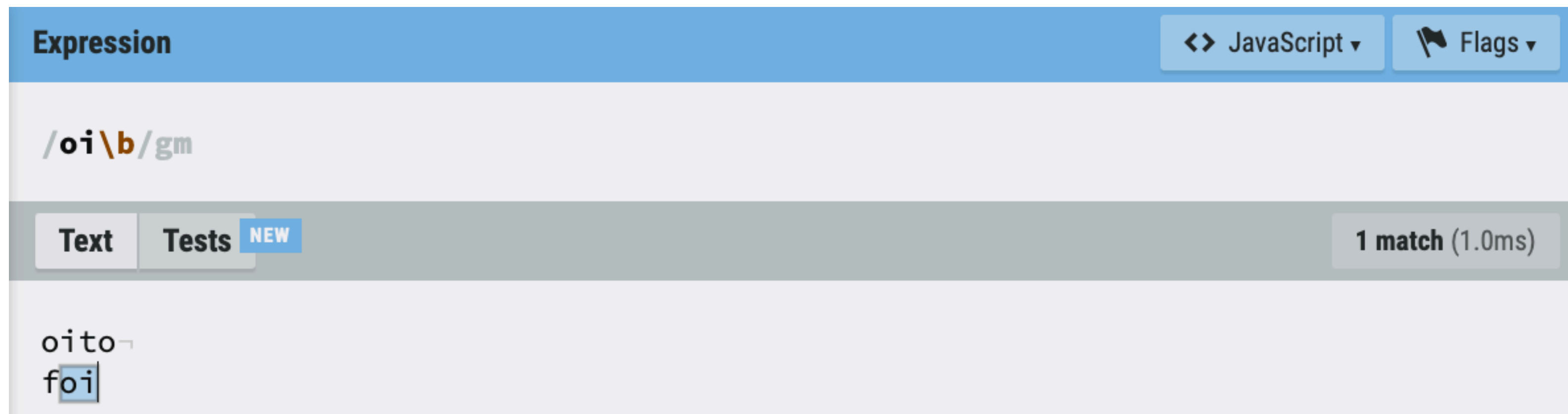
# Expressões Regulares

- Usando o **\b**, especificamos a fronteira de uma palavra
- O padrão **\boi** encontra o "oi" em "oito", mas não em "foi"



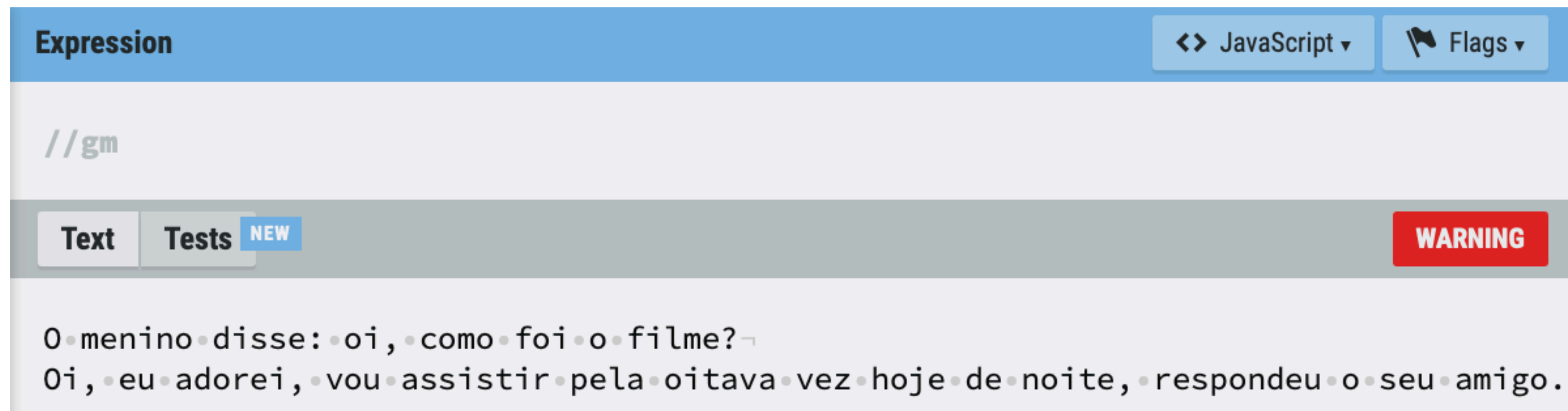
# Expressões Regulares

- Usando o **\b**, especificamos a fronteira de uma palavra
- O padrão **oi\b** encontra o "oi" em "foi", mas não em "oito"



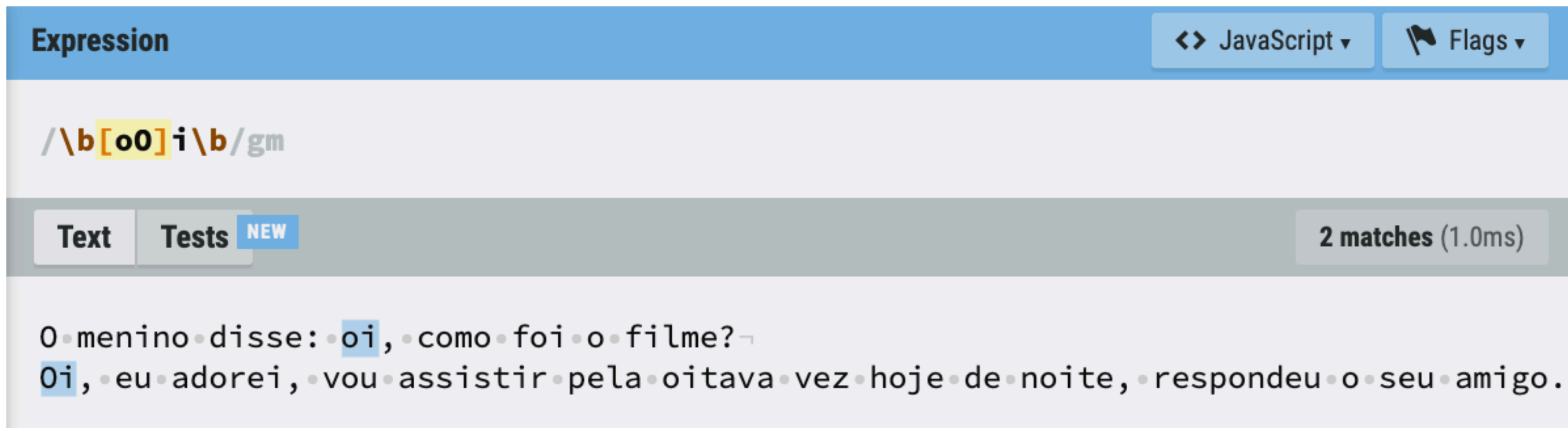
# Expressões Regulares

- Como criar um padrão que encontre todas as palavras “oi” no texto abaixo:



# Expressões Regulares

- Utilizando `\b[oO]i\b` conseguimos o resultado esperado:



The screenshot shows a web-based regular expression testing interface. At the top, there's a blue header with the word "Expression" on the left, and two buttons on the right: "<> JavaScript" and a flag icon followed by "Flags". Below the header, the regular expression `/\b[oO]i\b/gm` is entered in a text field. Underneath the text field is a grey bar containing three tabs: "Text", "Tests", and "NEW" (which is highlighted in blue). To the right of these tabs, it says "2 matches (1.0ms)". The main area below the grey bar displays two lines of text with matches highlighted in blue. The first line is "O menino disse: oi, como foi o filme?" and the second line is "Oi, eu adorei, vou assistir pela oitava vez hoje de noite, respondeu o seu amigo." The matches are "oi" in the first line and "Oi" in the second line.

```
Expression <> JavaScript Flags
```

```
/\b[oO]i\b/gm
```

Text Tests NEW 2 matches (1.0ms)

```
O menino disse: oi, como foi o filme?
Oi, eu adorei, vou assistir pela oitava vez hoje de noite, respondeu o seu amigo.
```

# Python

- Python possui o módulo **re** que nos permite trabalhar com expressões regulares
- O módulo oferece três funções principais para casamento de padrões usando expressões regulares



# Python

- `re.search()` - procura o padrão em qualquer parte de um string
- `re.search("linguagem", "processamento de linguagem natural")`
  - Se o padrão for encontrado, retorna um objeto do tipo Match
  - Se o padrão não for encontrado, retorna None

# Python

- `re.match()` - procura o padrão a partir do início de um string
- `re.match("linguagem", "processamento de linguagem natural")`
  - Retorna None
- `re.match("processamento", "processamento de linguagem natural")`
  - Retorna um objeto Match

# Python

- Podemos usar: `re.search("^processamento", "processamento de linguagem natural")` para ter o mesmo efeito do `match`

# Python

- `re.fullmatch()` - procura se todo o string casa com o padrão
- `re.fullmatch("linguagem", "processamento de linguagem natural")`
  - Retorna None
- `re.fullmatch("processamento de linguagem natural", "processamento de linguagem natural")`
  - Retorna um objeto Match

# Python

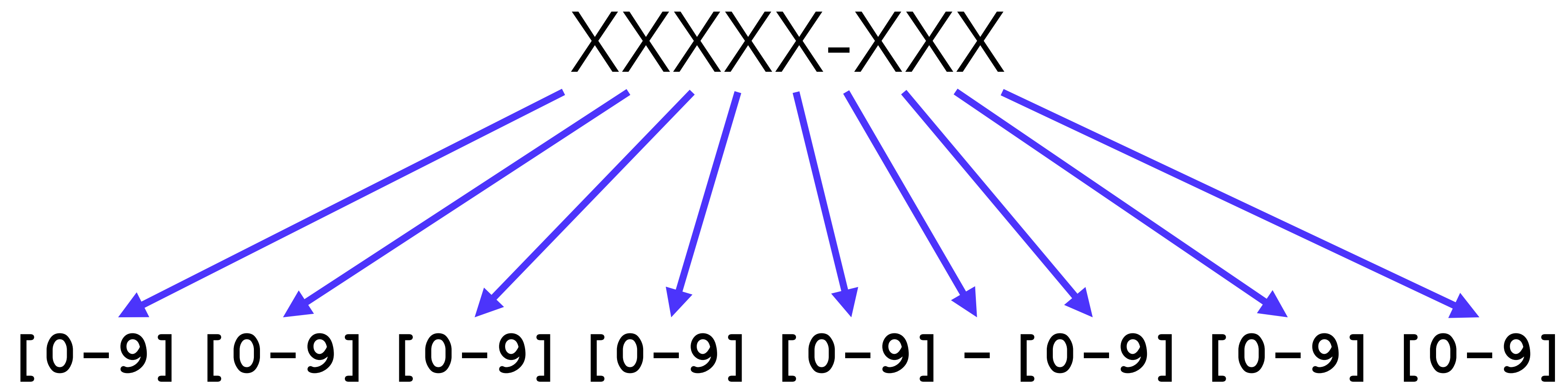
- Podemos usar: `re.search("^processamento de linguagem natural$", "processamento de linguagem natural")` para ter o mesmo efeito do `fullmatch`

# Python

- Expressões regulares são muito usadas para validação
- Por exemplo, para checar se um string corresponde a um número de telefone ou um email válido

# Python

- Vamos criar uma expressão regular para validar um CEP
- O CEP tem o formato XXXXX-XXX, onde X é um dígito



# Python

- O padrão encontrado ficou muito repetitivo
- Posso usar quantificadores **+**, **\*** ou **?** nesse caso? Não.
- Podemos especificar um número fixo de ocorrências utilizando **{n}**
  - Onde **n** é a quantidade de vezes que o caractere deve aparecer



# Python

- Assim, a expressão regular para o CEP pode ser definida como:
  - **[0-9]{5}-[0-9]{3}**

# Python

- Também podemos especificar intervalos de repetições usando a sintaxe **{n,m}**
  - **n** é a quantidade mínima e **m** a quantidade máxima

# Python - Exercício

- Vamos fazer uma validação para e-mails com os seguintes requisitos:
  - Um email é composto de usuário@domínio
  - Usuário é uma sequência com um ou mais caracteres que podem ser: letras, dígitos ou underline
  - Domínio é uma sequência com um ou mais caracteres que podem ser: letras, dígitos ou pontos.
    - Ele precisa terminar com “.X”, onde X é uma sequência de no mínimo 2 letras

# Python

- Exemplo:
  - E-mail: yuri@ci.ufpb.br
  - Usuário: yuri
  - Domínio: ci.ufpb.br

# Python

- Expressão regular:
  - Usuário: **[a-zA-Z0-9\_]+**
  - Domínio: **[a-zA-Z0-9\.] + \.[a-zA-Z]{2}[a-zA-Z]\***
  - Resposta:  
**[a-zA-Z0-9\_]+@[a-zA-Z0-9\.] + \.[a-zA-Z]{2}[a-zA-Z]\***

# Python

- O padrão de letras, dígitos e underline é tão comum que existe um atalho para ele
  - Basta usar **\w**
  - Equivalente a: **[a-zA-Z0-9\_]**
  - **\W** representa o oposto (tudo que não é letras, dígitos ou underline)

# Python

- Temos esses mesmo atalhos para o padrão de dígitos
  - **[0-9]** pode ser escrito como **\d**
  - **\D** casa com tudo que não é dígito

# Python

- Também existe um atalho especial para espaços em branco, que incluem tab, quebra de linha e o próprio espaço
  - **\s** representa espaços em branco
  - **\S** representa tudo que não é espaço em branco



# Python

- A módulo `re` ainda inclui a função `findall` que procura todas as ocorrências de um padrão em um texto
- Esta é uma poderosa ferramenta de busca que pode ser utilizada para buscar padrões complexos em grandes volumes de dados

# Python

- Dada a frase, “Ele foi cuidadosamente mas foi capturado rapidamente pela polícia” vamos procurar todos os advérbios nela
- Neste caso, vamos considerar que um advérbio é uma palavra que termina com “mente”
- `text = "Ele foi cuidadosamente mas foi capturado rapidamente pela polícia"`

```
re.findall("\w+mente", text)
```

# Python

- Todo padrão entre parênteses define um grupo
- O conteúdo de um grupo pode ser recuperado depois que o casamento do padrão é efetuado

```
text = "o produto custou R$ 80,00"  
match = re.search("R\$ (\d+,\d+)", text)  
  
print(match.group(1))
```

# Python

- Possuindo mais de um grupo, podemos acessá-los variando o parâmetro do método `.group()`

```
email_regex = "([a-zA-Z0-9_]+)(@[a-zA-Z0-9\.\.]{2}[a-zA-Z]*)"  
  
match = re.match(email_regex, "yuri@ci.ufpb.br")  
  
print(match.group(1))  
print(match.group(2))
```

# Python

- Com o `findall`, uma lista com o conteúdo dos grupos é retornada

```
text = "o produto custou R$ 80,00. O cliente pagou R$ 100,00 e recebeu R$ 20,00 de troco."  
match = re.findall("R\$ (\d+,\d+)", text)  
  
print(match)
```

# Python

- Para utilizar parênteses sem criar um grupo usamos a sintaxe **(?:padrão)**
- Por exemplo: **R\\$(?:\d+,\d+)**

# Python

- Utilizando a sintaxe **(?P<nome>)** podemos nomear os grupos para facilitar sua referência

```
text = "o produto custou R$ 80,00"  
match = re.search("R\$ (?P<preco>\d+,\d+)", text)  
  
print(match.group('preco'))
```

# Python

- O módulo `re` tem a função `sub()` que efetua substituição de strings
- Ela é definida assim: `re.sub(padrao, string, texto)`
- A ocorrência do `padrao` em `string` é substituída pelo parâmetro `texto`

```
text = "o produto custou R$ 80,00"  
result = re.sub("R\$ (\d+,\d+)", "XXXXX", text)  
  
print(result)
```



# Python

- É possível referenciar grupos na função **sub**
  - Para isso usamos **\x**, onde **x** é o número de grupo, ou **\g<nome\_do\_grupo>**
- Dada uma lista com nome e sobrenome, vamos substitui-los pelos e-mails de cada um
  - O e-mail é composto da primeira letra do primeiro nome, seguido do sobrenome e finalizado pelo domínio @email.com