ORIGAMID

REGULAR EXPRESSION

Regexp Seleção

Regexr

https://regexr.com/

Regular Expression

Regexp ou Regex são expressões utilizadas para realizarmos buscas / substituições de padrões em strings. Os padrões devem ser colocados entre `//`. Geralmente vamos utilizá-las nos métodos `.replace()` e `.split()`.

```
// Procura: a
const padraoRegexp = /a/;

const texto = 'JavaScript';
const novoTexto = texto.replace(padraoRegexp, 'B');
// BavaScript
```

> *Praticamente todas as linguagens possuem uma implementação de regexp*

Literal

Utilizar um caracter literal irá realizar uma busca específica deste caracter.

```
// Procura: J seguido de a, v e a
const regexp = /Java/;

'JavaScript'.replace(regexp, 'Type');
// TypeScript
```

Flag: g

As flags irão modificar como a expressão é interpretada. Uma das mais utilizadas é a `g`, que significa global, ou seja, retorne todos os resultados que estiverem dentro do padrão e não apenas o primeiro. A flag deve ser colocada no final da expressão.

```
// Procura: Todo a
const regexp = /a/g;

'JavaScript'.replace(regexp, 'i');
// JiviScript
```

Flag: i

Com o `i` informamos que devem ser ignoradas as diferenças entre maiúsculas e minúsculas. Isso significa que `/a/` irá buscar por `a e A`.

```
// Procura: Todo PE, Pe, pE e pe
const regexp = /Pe/gi;

'Perdeu perdido'.replace(regexp, 'Ba');
// Bardeu Bardido
```

Character Class

Se colocarmos os caracteres entre colchetes, estamos definindo uma classe. `/[ab]/` irá procurar por a ou por b.

```
// Procura: Todo a, A, i, I
const regexp = /[ai]/gi;

'JavaScript'.replace(regexp, 'u');
// JuvuScrupt
```

Character Class e Especiais

Podemos utilizar caracteres que não são alfanuméricos dentro da classe. Mas fique atento, pois existem diversos casos especiais para os mesmos.

```
// Procura: - ou .
const regexp = /[-.]/g;

'111.222-333-44'.replace(regexp, '');
// 11122233344
```

Um ou Outro

Combine caracteres literais com uma classe para buscarmos variações: `Ju[nl]ho` busca `Julho` ou `Junho`.

```
// Procura: B, seguido de r, a
// seguido de s ou z, seguido de i, l
const regexp = /Bra[sz]il/g;

'Brasil é com z: Brazil'.replace(regexp, 'Prazer');
// Prazer é com z: Prazer
```

De A à Z

O traço `-` dentro de `[]` pode servir para definirmos um alcance. `[A-Z]` irá buscar os caracteres de A à Z. `[0-9]` busca de 0 à 9. A tabela UNICODE é utilizada como referência para definir os caracteres dentro do alcance.

```
// Busca por itens de a à z
const regexp = /[a-z]/g;

'JavaScript é a linguagem.'.replace(regexp, '0');
// J000S00000 é 0 000000000.

// Busca por itens de a à z e A à Z
const regexp = /[a-zA-Z]/g;

'JavaScript é a linguagem.'.replace(regexp, '1');
// 1111111111 é 1 111111111.

// Busca por números de 0 à 9
const regexpNumero = /[0-9]/g;

'123.333.333-33'.replace(regexpNumero, 'X');
```

> https://unicode-table.com/en/

Negar

Utilizando o acento circunflexo podemos negar caracteres. Ou seja, pegue tudo que não seja `[^a]`

```
// Procura: tudo que não estiver entre a e z
const regexp = /[^a-z]/g;

'Brasil é com z: Brazil'.replace(regexp, ' ');
// rasil   com z   razil
```

Ponto

O ponto `.` irá selecionar qualquer caracter, menos quebras de linha.

```
// Procura: todos os caracteres menos quebra de linha
const regexp = /./g;

'JavaScript é a linguagem.'.replace(regexp, '0');
// 0000000000000000000000000
```

Escapar Especiais

Caracteres especiais como o ponto `.`, podem ser escapados utilizando a barra `\`, assim este não terá mais a sua função especial e será tratado como literal. Lista de caracteres especiais: `+*?^$\.[]{}()|/`

```
// Procura: todos os pontos
const regexp = /\./g;
const regexpAlternativa = /[.]/g;

'999.222.222.11'.replace(regexp, '-');
// 999-222-222-11
```

Word

O `\w` irá selecionar qualquer caracter alfanumérico e o underline.
É a mesma coisa que `[A-Za-z0-9_]`.

```
// Procura: todos os alfanuméricos
const regexp = /\w/g;

'Guarda-chuva R$ 23,00.'.replace(regexp, '-');
// ----------- -$ --,--.
```

Not Word

O `\W` irá selecionar tudo o que não for caracter alfanumérico e o underline. É a mesma coisa que `[^A-Za-z0-9_]`.

```
// Procura: o que não for caracter alfanuméricos
const regexp = /\W/g;

'Guarda-chuva R$ 23,00.'.replace(regexp, '-');
// Guarda-chuva-R--23-00-
```

Digit

O `\d` irá selecionar qualquer dígito. É a mesma coisa que `[0-9]`.

```
// Procura: todos os dígitos
const regexp = /\d/g;

'+55 (21) 2222-2222'.replace(regexp, 'X');
// +XX (XX) XXXX-XXXX.
```

Not Digit

O `\D` irá selecionar tudo que não for dígito. É a mesma coisa que `[^0-9]`.

```
// Procura: o que não for dígito
const regexp = /\D/g;

'+55 (21) 2222-2222'.replace(regexp, '');
// 552122222222
```

Whitespace

O `\s` irá selecionar qualquer espaço em branco, isso inclui espaços, tabs, quebra de linhas.

```
// Procura: espaços em branco
const regexp = /\s/g;

'+55 (21) 2222-  2222  '.replace(regexp, '');
// +55(21)2222-2222
```

Not Whitespace

O `\S` irá selecionar qualquer caracter que não for espaço em branco.

```
// Procura: o que não for espaço em branco
const regexp = /\S/g;

'+55 (21) 2222-  2222  '.replace(regexp, '');
// XXX XXXX XXXXX  XXXX
```

> `/[\s\S]/g` irá selecionar tudo.

Quantificador

É possível selecionar caracteres seguidos, como `/bbb/g` irá selecionar apenas `bbb`. Com as chaves podemos indicar a repetição `/b{3}/g`. Agora ele está fazendo uma seleção completa e não caracter por caracter.

```
// Procura: 4 a's seguidos
const regexp = /aaaa/g;
const regexpAlt = /a{4}/g;

'Vaaaai ali por favor.'.replace(regexp, 'a');
// Vai ali por favor.
```

Quantificador Min e Max

Podemos informar o min e max do quantificador `/a{2,4}/` vai selecionar quando aparecer `a` duas vezes ou até 4 vezes.
`/a{2,}/` irá selecionar quando se repetir duas ou mais vezes.

```
// Procura: dígitos seguidos de 2 à 3
const regexp = /\d{2,3}/g;

'222.333.222.42'.replace(regexp, 'X');
// X.X.X.X

// Procura: letras seguidos com 1 caracter ou mais
const regexpLetras = /\w{1,}/g;

'A melhor linguagem é JavaScript'.replace(regexpLetras, 'X');
// X X X é X
```

Mais +

O sinal de + significa que devemos selecionar quando existir pelo menos uma ou mais ocorrências.

```
// Procura: dígitos em ocorrência de um ou mais
const regexp = /\d+/g;

'222.333.222.42'.replace(regexp, 'X');
// X.X.X.X

// Procura: Começa com d, seguido por uma ou mais letras.
const regexpLetras = /d\w+/g;

'Dígitos, dados, desenhos, Dito, d'.replace(regexpLetras, 'X');
// Dígitos, X, X, Dito, d
```

Asterisco *

O sinal `*` significa que devemos selecionar quando existir 0 ou mais ocorrências.

```
// Procura: Começa com d, seguido por zero ou mais letras.
const regexp = /d\w*/g;

'Dígitos, dados, desenhos, Dito, d'.replace(regexp, 'X');
// Dígitos, X, X, Dito, X
```

Opcional ?

O sinal `?` significa que o caracter é opcional, pode ou não existir.

```
// Procura: Por regex com p opcional
const regexp = /regexp?/g;

'Qual é o certo, regexp ou regex?'.replace(regex, 'Regular Expression');
// Qual é o certo, Regular Expression ou Regular Expression?
```

Alternado |

O sinal `|` irá selecionar um ou outro. `java|php`

```
// Procura: java ou php (case insensitive)
const regexp = /java|php/gi;

'PHP e Java são linguagens diferentes'.replace(regexp, 'X');
// X e X são linguagens diferente
```

Word Boundary \b

O sinal `\b` irá indicar que pretendemos fazer uma seleção que deve ter início e fim de não caracteres `\w`.

```
// Procura: java (case insensitive)
const regexp = /java/gi;
'Java não é JavaScript.'.replace(regexp, 'X');
// X não é XScript.

// Procura: java (case insensitive)
const regexpBoundary = /\bjava\b/gi;
'Java não é JavaScript.'.replace(regexpBoundary, 'X');
// X não é JavaScript.

// Procura: Dígitos em sequência, que estejam isolados
const regexpDigito = /\b\d+\b/gi;
'O Restaurante25 na Rua 3, custa R$ 32,00'.replace(regexDigito,
'X');
// O Restaurante25 na Rua X, custa R$ X,X

'11_22 33-44 55é66 77e88'.replace(regexpDigito, 'X');
// 11_22 X-X XéX 77e88
```

Not Word Boundary \B

É o contrário do `\b`.

```
const regexpDigito = /\B\d+\B/gi;

'11_22 33-44 55é66 77e88'.replace(regexpDigito, 'X');
// 1X_X2 33-44 55é66 7XeX8
```

Anchor Beginning

Com o `^` é possível informar que a busca deve ser iniciada no início da linha.

```
// Procura: sequência de alfanuméricos
// no início da linha.
const regexp = /^\w+/g;

`andre@origamid.com
contato@origamid.com`.replace(regexp, 'X');
// X@origamid.com
// contato@origamid.com
```

Anchor End

Com o `$` é possível informar que a busca deve ser iniciada no final da linha.

```
// Procura: sequência de alfanuméricos
// no final da linha.
const regexp = /\w+$/g;

`andre@origamid.com
contato@origamid.com`.replace(regexp, 'X');
// andre@origamid.com
// contato@origamid.X
```

Flag: m

Com a flag `m` de multiline, podemos informar que a busca de início `^` e final `$` de linha devem ocorrer em todas as linhas disponíveis.

```
// Procura: sequência de alfanuméricos
// no final da linha.
const regexp = /\w+$/gm;

`andre@origamid.com
contato@origamid.com`.replace(regexp, 'X');
// andre@origamid.X
// contato@origamid.X

// Procura: sequência de alfanuméricos
// no início da linha.
const regexp = /^\w+/gm;

`andre@origamid.com
contato@origamid.com`.replace(regexp, 'X');
// X@origamid.com
// X@origamid.com
```

Line Feed \n

O `\n` irá selecionar o final de uma linha, quando criamos uma nova.

```
const regexp = /\n/g;

`andre@origamid.com\ncontato@origamid.com`.replace(regexp, '---
');
// andre@origamid.com---contato@origamid.com

`andre@origamid.com
contato@origamid.com`.replace(regexp, 'X');
// andre@origamid.com---contato@origamid.com
```

> `\t` *seleciona tabs*

Unicode \u

O `\u` irá selecionar o respectivo caracter unicode, de acordo com o código passado em `\uXXXX`. Ex: `\u0040` seleciona o `@`.

```
// Procura: @ ou ©
const regexp = /\u0040|\u00A9/g;

'andre@origamid.com ©'.replace(regexp, '---');
// andre---origamid.com ---
```