Regular Expressions / JavaScript

Usar o método test

Expressões regulares são usadas em linguagens de programação para encontrar e extrair partes de strings. Cria-se padrões que ajudam a encontrar tais partes.

Se você quiser encontrar a palavra `the` na string `The dog chased the cat`, você poderia usar a seguinte expressão regular: `/the/`. Note que aspas não são necessárias ao redor da expressão regular.

O JavaScript oferece múltiplas maneiras de usar *regexes*. Uma dessas maneiras é com o método `.test()`. O método `.test()` aplica a regex à string dentro dos parênteses e retorna `true` caso encontre o padrão ou `false` caso contrário.

```
let testStr = "freeCodeCamp";
let testRegex = /Code/;
testRegex.test(testStr);
```

O método `test` retorna `true` aqui.

Exercicio = Aplique a regex `myRegex` na string `myString` usando o método `.test()`.

```
let myString = "Hello, World!";

let myRegex = /Hello/;

let result = myRegex.test(myString); // Altere esta linha
```

Criar correspondência de strings literais

No desafio anterior, você usou a expressão regular `/Hello/` para procurar a palavra `Hello`. Esta regex buscou a string `Hello` literalmente. No exemplo abaixo há outra busca literal, dessa vez pela string `Kevin`:

```
let testStr = "Hello, my name is Kevin.";
let testRegex = /Kevin/;
testRegex.test(testStr);
```

Essa chamada a `test` retornará `true`.

Qualquer outra forma de escrever `Kevin` não funcionará. Por exemplo, a regex `/Kevin/` não encontrará nem `kevin` e nem `KEVIN`.

```
let wrongRegex = /kevin/;
wrongRegex.test(testStr);
```

`test` retornará `false`.

Você verá como encontrar estas outras formas em alguns desafios futuros.

Obs : acho que tem que ser escrito maiúsculas e minúsculas iguais .

Exercicio = Complete a regex `waldoRegex` para encontrar `"Waldo"` na string `waldoIsHiding` de forma literal.

```
let waldoIsHiding = "Somewhere Waldo is hiding in this text.";

let waldoRegex = /Waldo/; // Altere esta linha

let result = waldoRegex.test(waldoIsHiding);
```

Buscar uma string literal com diferentes possibilidades

Ao usar regexes como `/coding/`, você pode procurar pelo padrão `coding` em strings.

Isso funciona com strings únicas, mas é limitado a apenas um padrão. Você pode procurar por múltiplos padrões usando o operador de `alternation`, ou `OR`: `|`.

Este operador funciona para buscar padrões à esquerda e à direita dele. Por exemplo, se você quiser encontrar as strings `yes` ou `no`, a regex que você quer é `/yes|no/`.

Você pode também procurar por mais de dois padrões com este operador. É possível fazer isso ao adicionar mais instâncias do operador seguido do padrão desejado: `/yes|no|maybe/`.

Exercicio = Complete a regex `petRegex` para encontrar os pets `dog`, `cat`, `bird`, ou `fish`.

```
let petString = "James has a pet cat.";

let petRegex = /change|dog|cat|bird|fish/; // Altere esta linha

let result = petRegex.test(petString);
```

Ignorar maiúsculas e minúsculas ao buscar

Até agora você escreveu regexes para encontrar strings literais. Mas, às vezes, você pode querer encontrar caixas diferentes.

Caixa (-alta ou -baixa) é a diferença entre letras maiúsculas e minúsculas. São exemplos de caixa alta: `A`, `B` e `C`. `a`, `b` e `c` são exemplos de caixa baixa.

Você pode encontrar ambas as caixas usando algo que chamamos de *flag*. Existem várias flags, mas agora nós queremos a flag que ignora a caixa - a flag `i`. Para usá-la é só colocar ao fim da regex. Por exemplo, escrever `/ignorecase/i` é uma forma. Essa regex pode encontrar as strings `ignorecase`, `igNoreCase` e `IgnoreCase` (e todas as outras combinações de maiúsculas e minúsculas).

Exercicio = Escreva uma regex `fccRegex` que encontre `freeCodeCamp`, não importa em que caixa esteja. A regex não deve buscar abreviações ou variações com espaços.

```
let myString = "freeCodeCamp";

let fccRegex = /freecodecamp/i; // Altere esta linha
```

```
let result = fccRegex.test(myString);
```

Extrair resultados

Até agora, você só tem verificado se existe ou não um padrão dentro de uma string. Você também pode extrair os resultados encontrados por meio do método `.match()`.

Para usar o método `.match()`, aplique o método em uma string e passe a regex dentro dos parênteses.

Exemplo:

```
"Hello, World!".match(/Hello/);
let ourStr = "Regular expressions";
let ourRegex = /expressions/;
ourStr.match(ourRegex);
```

Aqui, o primeiro `match` retorna `["Hello"]` e, o segundo, `["expressions"]`.

Note que o método `.match` se usa de forma "contrária" ao método `.test` que você usou até então:

```
'string'.match(/regex/);
/regex/.test('string');
```

Exercicio = Aplique o método `.match()` para extrair a string `coding`.

```
let extractStr = "Extract the word 'coding' from this string.";

let codingRegex = /coding/; // Altere esta linha

let result = extractStr.match(codingRegex); // Altere esta linha
```

Encontrar mais do que o primeiro resultado

Até agora você foi capaz apenas de extrair ou buscar um resultado de uma vez.

```
let testStr = "Repeat, Repeat, Repeat";
let ourRegex = /Repeat/;
testStr.match(ourRegex);
```

`match` retorna `["Repeat"]` aqui.

Para buscar ou extrair um padrão além do primeiro resultado, você pode usar a flag `g` (de "global").

```
let repeatRegex = /Repeat/g;
testStr.match(repeatRegex);
```

Aqui, `match` retorna o valor `["Repeat", "Repeat", "Repeat"]`

Exercicio = Usando a regex `starRegex`, encontre e extraia ambas ocorrências da palavra `Twinkle` da string `twinkleStar`.

Observação:
você pode usar múltiplas flags em uma regex: `/search/gi`

```
let twinkleStar = "Twinkle, twinkle, little star";

let starRegex = /Twinkle/ig; // Altere esta linha

let result = twinkleStar.match(starRegex); // Altere esta linha
```

Encontrar qualquer coisa com o caractere curinga

Haverá vezes em que você não saberá (ou não precisará saber) quais caracteres exatamente farão parte das suas regexes. Pensar em todas as

palavras que capturariam, digamos, um erro ortográfico levaria muito tempo. Por sorte, você pode economizar tempo usando o caractere curinga: `.`

O caractere curinga `.` captura qualquer caractere. O curinga também pode ser chamado de `ponto`. Você pode usar o curinga como qualquer outro caractere na regex. Por exemplo, se você quiser encontrar `hug`, `huh`, `hut` ou `hum`, você pode usar a regex `/hu./` para capturar todas as quatro palavras.

```
let humStr = "I'll hum a song";
let hugStr = "Bear hug";
let huRegex = /hu./;
huRegex.test(humStr);
huRegex.test(hugStr);
```

As duas chamadas a `test` retornam `true`.

Exercicio = Complete a regex `unRegex` para que ela encontre as strings `run`, `sun`, `fun`, `pun`, `nun` e `bun`. A regex deve usar o caractere curinga.

```
let exampleStr = "Let's have fun with regular expressions!";

let unRegex = /.un/; // Altere esta linha

let result = unRegex.test(exampleStr);
```

Capturar um único caractere com múltiplas possibilidades

Você aprendeu a capturar padrões literais (`/literal/`) e usar o caractere curinga (`/./`). Eles são os extremos das expressões regulares: um encontra o texto exato e o outro captura qualquer coisa. Existem formas de balancear esses extremos.

Você pode ter alguma flexibilidade ao procurar um padrão literal usando *classes de caracteres*. Classes de caracteres permitem a definição de grupos de caracteres que você quer capturar ao colocá-los entre colchetes: `[` e `]`.

Por exemplo, se você quiser encontrar `bag`, `big` e `bug` mas não `bog`. Você pode escrever a regex `/b[aiu]g/` para isso. `[aiu]` é a classe de caracteres que só capturará `a`, `i` ou `u`.

```
let bigStr = "big";
let bagStr = "bag";
let bugStr = "bug";
let bogStr = "bog";
let bgRegex = /b[aiu]g/;
bigStr.match(bgRegex);
bagStr.match(bgRegex);
bugStr.match(bgRegex);
bogStr.match(bgRegex);
```

As quatro chamadas a `match` retornarão os seguintes valores, nesta ordem: `["big"]`, `["bag"]`, `["bug"]` e `null`.

Exercicio = Use classe de caracteres de vogais (`a`, `e`, `i`, `o`, `u`) na sua regex `vowelRegex` para encontrar todas as vogais na string `quoteSample`.

Observação: você quer encontrar tanto maiúsculas quanto minúsculas.

```
let quoteSample =

  "Beware of bugs in the above code; I have only proved it
correct, not tried it.";

let vowelRegex = /[aeiou]/gi; // Change this line

let result = quoteSample.match(vowelRegex); // Change this
line
```

Capturar letras do alfabeto

Você viu como pode usar *conjuntos de caracteres* para especificar um grupo de caracteres para capturar. Mas você precisaria escrever muito para definir uma classe larga como, por exemplo, para capturar todas as letras do alfabeto. Felizmente há uma maneira de fazer com que elas fiquem pequenas e simples.

Você pode usar um hífen (-) para definir um intervalo de caracteres para capturar dentro de uma classe.

Por exemplo, para encontrar letras minúsculas de `a` a `e`, você pode escrever `[a-e]`.

```
let catStr = "cat";
let batStr = "bat";
let matStr = "mat";
let bgRegex = /[a-e]at/;
catStr.match(bgRegex);
batStr.match(bgRegex);
matStr.match(bgRegex);
```

As três chamadas a `match` retornam, na ordem, os valores: `["cat"]`, `["bat"]` e `null`.

Exercicio = Capture todas as letras na string `quoteSample`.

Observação: você quer encontrar tanto maiúsculas quanto minúsculas.

```
let quoteSample =

"The quick brown fox jumps over the lazy dog.";

let alphabetRegex = /[a-z]/gi; // Altere esta linha

let result = quoteSample.match(alphabetRegex); // Altere
esta linha
```

Capturar números e letras do alfabeto

O uso do hífen (-) para capturar um intervalo de caracteres não é limitado a letras. Ele também funciona para capturar intervalos de números.

Por exemplo, `/[0-5]/` encontra qualquer número entre `0` e `5`, incluindo ambos `0` e `5`.

E também é possível combinar intervalos de letras e números em uma única classe de caracteres.

```
let jennyStr = "Jenny8675309";
let myRegex = /[a-z0-9]/ig;
jennyStr.match(myRegex);
```

Exercicio = Escreva uma única regex que encontra letras entre `h` e `s` e, também, números entre `2` e `6`. Lembre-se de incluir as flags necessárias na regex.

```
let quoteSample = "Blueberry 3.141592653s are delicious.";

let myRegex = /[h-s2-6]/ig; // Altere esta linha

let result =quoteSample.match (myRegex); // Altere esta
linha
```

Capturar caracteres não especificados

Até agora você aprendeu a criar classes de caracteres para capturar caracteres específicos, mas você também pode usá-las para capturar caracteres ausentes nelas. Esse tipo de classe de caracteres é chamada *classe de caracteres negada*.

Para criar uma classe de caracteres negada, você só precisa colocar um acento circunflexo (^) depois do colchete de abertura e à esquerda dos caracteres que você quer evitar.

Por exemplo, `/[^aeiou]/gi` encontra todos os caracteres que não são vogais. Observe que caracteres como `.`, `!`, `[`, `@`, `/` e espaços em branco são capturados - a classe de vogais negada apenas exclui os caracteres que são vogais.

Exercicio = Crie uma única regex que captura todos os caracteres que não são números ou vogais. Lembre-se de incluir as flags necessárias na regex.

```
let quoteSample = "3 blind mice.";

let myRegex = /[^aeiou0-9]/ig; // Altere esta linha

let result = quoteSample.match(myRegex); // Altere esta
linha
```

Capturar caracteres que aparecem uma ou mais vezes seguidas

Às vezes você precisa capturar um caractere, ou grupo de caracteres, que aparece uma ou mais vezes seguidas. Ou seja, que aparecem pelo menos uma vez e podem se repetir.

Você pode usar o caractere `+` para verificar se é o caso. Lembre-se que o caractere ou padrão precisa repetir-se consecutivamente. Ou seja, um atrás do outro.

Por exemplo, `/a+/g` encontra um resultado na string `abc` e retorna `["a"]`. Mas o `+` também faz com que encontre um único resultado em `aabc` e retorne `["aa"]`.

Se a string fosse `abab`, a operação retornaria `["a", "a"]` porque entre os dois `a` há um `b`. Por fim, se não houvesse nenhum `a` na string, como em `bcd`, nada seria encontrado.

Exercicio = Você quer capturar as ocorrências de `s` quando acontecer uma ou mais vezes em `Mississippi`. Escreva uma regex que use o caractere `+`.

```
let difficultSpelling = "Mississippi";

let myRegex = /s+/gi; // Altere esta linha

let result = difficultSpelling.match(myRegex);
```

Capturar caracteres que aparecem zero ou mais vezes seguidas

O último desafio fez uso do caractere `+` para buscar caracteres que ocorrem uma ou mais vezes. Existe um outro caractere que permite buscar zero ou mais ocorrências de um padrão.

O caractere usado para isso é o asterisco: `*`.

```
let soccerWord = "gooooooooal!";
let gPhrase = "gut feeling";
let oPhrase = "over the moon";
let goRegex = /go*/;
soccerWord.match(goRegex);
```

```
gPhrase.match(goRegex);
oPhrase.match(goRegex);
```

As três chamadas a `match` retornam, na ordem, os valores: `["goooooooo"]`, `["g"]` e `null`.

Exercicio = Neste desafio, a string `chewieQuote` recebeu o valor `Aaaaaaaaaaaaaaarrrgh!` por trás dos panos. Escreva uma regex, `chewieRegex`, que usa o caractere `*` para capturar um `A` maiúsculo seguido imediatamente de zero ou mais `a` minúsculos em `chewieQuote`. A regex não precisa de flags ou de classes de caracteres. Ela também não deve capturar nenhuma outra parte da string.

```
let chewieQuote = "Aaaaaaaaaaaaaaarrrgh!";
let chewieRegex = /Aa*/; // Change this line
let result = chewieQuote.match(chewieRegex);
```

Encontrar caracteres com captura preguiçosa

Em expressões regulares, uma captura *gananciosa* encontra a parte mais longa o possível de uma string em que a regex atua e a retorna como resultado. A alternativa se chama captura *preguiçosa* e ela encontra o menor pedaço o possível de uma string que satisfaz a regex.

Você pode aplicar a regex `/t[a-z]*i/` à string `"titanic"`. Essa regex é basicamente um padrão que começa com `t`, termina com `i` e tem algumas letras no meio delas.

Expressões regulares são gananciosas por padrão, então o resultado seria `["titani"]`. Ou seja, a maior string o possível que atende ao padrão é encontrada.

Mas você pode usar o caractere `?` para torná-la preguiçosa. Aplicar a regex adaptada `/t[a-z]*?i/` à string `"titanic"` retorna `["ti"]`.

Observação: ler HTML com expressões regulares deve ser evitado, mas procurar uma string HTML usando expressões regulares é perfeitamente aceitável.

Exercicio = Arrume a regex `/<.*>/` para que retorne a tag HTML `<h1>` mas não a linha `"<h1>Winter is coming</h1>"`. Lembre-se de que o caractere curinga `.` em uma expressão regular captura qualquer caractere.

```
let text = "<h1>Winter is coming</h1>";
let myRegex = /<.*?>/; // Altere esta linha
let result = text.match(myRegex);
```

Encontrar um ou mais criminosos em uma caçada

Hora de testar as suas novas habilidades de regex. Um grupo de criminosos escapou da prisão, mas não sabemos quantos. No entanto, sabemos que eles ficam juntos quando estão no meio de outras pessoas. E você ficou responsável por encontrar todos eles.

Vamos revisitar maneiras de executar essa tarefa:

A regex `/z+/` encontra a letra `z` quando ela aparece uma ou mais vezes seguidas. Ela encontra resultados em todas essas strings:

```
"z"
"zzzzzz"
"ABCzzzz"
"zzzzABC"
"abczzzzzzzzzzzzzzzzzzzzzabc"
```

Mas ela não encontra nada nessas strings, que não possuem `z` em lugar nenhum:

```
""
"ABC"
```

```
"abcabc"
```

Exercicio = Escreva uma regex gananciosa que encontra uma ou mais criminosos em um grupo de pessoas. Um criminoso pode ser identificado pela letra maiúscula `C`.

```
let reCriminals = /C+/; // Altere esta linha
```

Encontrar padrões ao início da string

Desafios anteriores mostraram que expressões regulares podem ser usadas para capturar um número de resultados. Elas também podem ser usadas para procurar em posições específicas de strings.

Mais cedo você usou o circunflexo (^) em classes de caracteres para procurar caracteres que não devem ser capturados, como em `[^caracteresQueNãoQueremos]`. Quando usados fora de classes de caracteres, o circunflexo serve para buscar a partir do começo de strings.

```
let firstString = "Ricky is first and can be found.";
let firstRegex = /^Ricky/;
firstRegex.test(firstString);
let notFirst = "You can't find Ricky now.";
firstRegex.test(notFirst);
```

A primeira chamada a `test` retorna `true` enquanto a segunda retorna `false`.

Exercicio = Use o circunflexo em uma regex para encontrar `Cal`, mas apenas no começo da string `rickyAndCal`.

```
let rickyAndCal = "Cal and Ricky both like racing.";

let calRegex = /^Cal/; // Altere esta linha

let result = calRegex.test(rickyAndCal);
```

Encontrar padrões ao final da string

No desafio anterior, você aprendeu a usar o circunflexo para capturar padrões no início de strings. Há também uma maneira de buscar padrões no fim de strings.

Se você colocar um cifrão, `$`, no fim da regex, você pode buscar no fim de strings.

```
let theEnding = "This is a never ending story";
let storyRegex = /story$/;
storyRegex.test(theEnding);
let noEnding = "Sometimes a story will have to end";
storyRegex.test(noEnding);
```

A primeira chamada a `test` retorna `true` enquanto a segunda retorna `false`.

Exercicio = Use o cifrão (`$`) para capturar a string `caboose` no fim da string `caboose`.

```
let caboose = "The last car on a train is the caboose";

let lastRegex = /caboose$/; // Altere esta linha

let result = lastRegex.test(caboose);
```

Capturar todas as letras e números

Ao escrever `[a-z]` você foi capaz de capturar todas as letras do alfabeto. Essa classe de caracteres é tão comum que existe uma forma reduzida de escrevê-la. Mas essa forma inclui alguns caracteres a mais.

Em JavaScript, você pode usar `\w` para capturar todas as letras do alfabeto. Isso é equivalente à classe de caracteres `[A-Za-z0-9_]`. Ela captura números e letras, tanto maiúsculas quanto minúsculas. Note que o underline (_) também é incluído nela.

```
let longHand = /[A-Za-z0-9_]+/;
let shortHand = /\w+/;
let numbers = "42";
let varNames = "important_var";
longHand.test(numbers);
shortHand.test(numbers);
longHand.test(varNames);
shortHand.test(varNames);
```

As quatro chamadas a `test` retornam `true`.

Essas formas reduzidas de classes de caracteres podem ser chamadas de *atalhos*.

Exercicio= o atalho `\w` para contar o número de caracteres alfanuméricos em várias strings.

```
let quoteSample = "The five boxing wizards jump quickly.";
let alphabetRegexV2 = /\w/g;// Altere esta linha
let result = quoteSample.match(alphabetRegexV2);
```

Capturar tudo exceto letras e números

Você aprendeu que você pode usar um atalho para capturar alfanuméricos `[A-Za-z0-9_]` usando `\w`. Você pode querer capturar exatamente o oposto disso.

Você pode capturar não alfanuméricos usando `\W` ao invés de `\w`. Observe que o atalho usa uma maiúscula. Este atalho é o mesmo que escrever `[^A-Za-z0-9_]`.

```
let shortHand = /\W/;
let numbers = "42%";
let sentence = "Coding!";
numbers.match(shortHand);
sentence.match(shortHand);
```

A primeira chamada a `match` retorna `["%"]` enquanto o segundo retorna `["!"]`.

Exercicio = Use o atalho `\W` para contar o número de caracteres não alfanuméricos em várias strings.

```
let jujuO1 = "Pack my box with five dozen liquor jugs.";
let jujuO2 = "How vexingly quick daft zebras jump!";
let jujuO3 = "123 456 7890 ABC def GHI jkl MNO pqr STU vwx YZ.";
let quoteSample = "The five boxing wizards jump quickly.";
let nonAlphabetRegex = /\W/gi; // Altere esta linha
let result = quoteSample.match(nonAlphabetRegex).length;
jujuO1.match(nonAlphabetRegex).length;
jujuO2.match(nonAlphabetRegex).length;
jujuO3.match(nonAlphabetRegex).length;
```

Capturar todos os números

Você aprendeu atalhos para padrões comuns de strings como alfanuméricos. Outro padrão comum é o de apenas dígitos ou números.

O atalho para procurar caracteres numéricos é `\d`, com um `d` minúsculo. Esse atalho é o mesmo que `[0-9]`, que serve para procurar qualquer dígito de zero a nove.

Exercicio = Use o atalho `\d` para contar quantos dígitos existem em títulos de filmes. Números por extenso, como "seis" em vez de 6, não contam.

```
let movieName = "2001: A Space Odyssey";

let numRegex = /\d/ig; // Altere esta linha

let result = movieName.match(numRegex).length;
```

Capturar tudo exceto números

O último desafio mostrou como procurar dígitos usando o atalho `\d` com um `d` minúsculo. Você também pode procurar não dígitos usando um atalho semelhante que usa um `D` maiúsculo.

O atalho para procurar não dígitos é `\D`. Esse atalho é o mesmo que `[^0-9]`, que serve para procurar qualquer caractere que não seja um dígito de zero a nove.

Exercicio = Use o atalho `\D` para contar quantos não dígitos existem em títulos de filmes.

```
let movieName = "2001: A Space Odyssey";

let noNumRegex = /\D/gi; // Altere esta linha

let result = movieName.match(noNumRegex).length;
```

Restringir nomes de usuário possíveis

Nomes de usuário (usernames) são usados em toda a Internet. São o que fazem com que tenham uma identidade única em seus sites favoritos.

Você precisa verificar todos os usernames em um banco de dados. Existem algumas regras que os usuários precisam seguir quando criam os seus usernames.

1. Nomes de usuário só podem conter caracteres alfanuméricos.
2. Os números, se algum, precisam estar no fim da string. Pode haver zero ou mais números. Usernames não podem começar com números.
3. As letras podem ser maiúsculas ou minúsculas.
4. O tamanho de nomes de usuários precisa ser pelo menos dois. Um username de dois caracteres só pode conter letras.

Exercicio = Modifique a regex `userCheck` para que inclua as regras listadas.

```
let username = "JackOfAllTrades";
let userCheck = /^[a-z][a-z]+\d*$|^[a-z]\d\d+$/i; // Altere esta linha
let result = userCheck.test(username);
```

Capturar espaços em branco

Os desafios até agora cobriram a captura de letras do alfabeto e números. Você também pode capturar espaços em branco e os espaços entre as letras.

Você pode usar o atalho `\s` com um `s` minúsculo para capturar espaços em branco. Este atalho não captura apenas espaços em branco como também retorno de carro, tabulações, feeds de formulário e quebras de linha. O atalho é equivalente à classe de caracteres `[ \r\t\f\n\v]`.

```
let whiteSpace = "Whitespace. Whitespace everywhere!"
let spaceRegex = /\s/g;
whiteSpace.match(spaceRegex);
```

`match` retorna `[" ", " "]` aqui.

Exercicio = Mude a regex `countWhiteSpace` para que capture múltiplos espaços em branco em strings.

```
let sample = "Whitespace is important in separating words";
let countWhiteSpace = /\s/g; // Altere esta linha
let result = sample.match(countWhiteSpace);
```

Capturar caracteres além do espaço

Você aprendeu a procurar por espaço em branco usando `\s` com um `s` minúsculo. Você também pode buscar tudo exceto espaços em branco.

Busque não espaços em branco usando `\S` com um `s` maiúsculo. Este atalho não captura espaços em branco, retorno de carro, tabulações, feeds de formulário ou quebras de linha. O atalho é equivalente à classe de caracteres `[^ \r\t\f\n\v]`.

```
let whiteSpace = "Whitespace. Whitespace everywhere!"
let nonSpaceRegex = /\S/g;
whiteSpace.match(nonSpaceRegex).length;
```

O valor retornado pelo método `.length` aqui é `32`.

Exercicio = Modifique a regex `countNonWhiteSpace` para que encontre tudo exceto espaços em branco em uma string.

```
let sample = "Whitespace is important in separating words";

let countNonWhiteSpace = /\S/g; // Altere esta linha

let result = sample.match(countNonWhiteSpace);
```

Especificar o número de capturas

Lembre-se de que você pode usar o sinal de `+` para procurar por uma ou mais ocorrências e o asterisco `*` para procurar por zero ou mais ocorrências. Eles são convenientes, mas às vezes você precisa capturar um número exato de caracteres.

Você pode especificar um número mínimo e um máximo de capturas com *especificadores de quantidade*. Para usar especificadores de quantidade, usa-se chaves: `{` e `}`. Você pode especificar os dois números dentro delas para restringir as capturas.

Por exemplo, se você quiser encontrar a letra `a` de `3` a `5` vezes na string `ah`, você pode escrever a regex `/a{3,5}h/`.

```
let A4 = "aaaah";
let A2 = "aah";
let multipleA = /a{3,5}h/;
multipleA.test(A4);
multipleA.test(A2);
```

A primeira chamada a `test` retorna `true` enquanto a segunda retorna `false`.

Exercicio = Altere a regex `ohRegex` para que capture a frase `Oh no`, mas apenas quando nela houver de `3` a `6` letras `h`'s.

```
let ohStr = "Ohhh no";

let ohRegex = /Oh{3,6}\sno/; // Altere esta linha

let result = ohRegex.test(ohStr);
```

Especificar apenas o mínimo de capturas

Você pode especificar um número mínimo e um máximo de capturas com chaves. Haverá vezes que você precisará especificar um mínimo mas não um máximo.

Para fazer isso, apenas escreva o número seguido de uma vírgula dentro das chaves.

Por exemplo, para capturar pelo menos `3` vezes a letra `a` na string `hah` você pode escrever a regex `/ha{3,}h/`.

```
let A4 = "haaaah";
let A2 = "haah";
let A100 = "h" + "a".repeat(100) + "h";
let multipleA = /ha{3,}h/;
multipleA.test(A4);
multipleA.test(A2);
```

```
multipleA.test(A100);
```

As três chamadas a `test` acima retornam, na ordem, os valores: `true`, `false` e `true`.

Exercicio = Modifique a regex `haRegex` para que capture quatro ou mais `z`s na string `Hazzah`.

```
let haStr = "Hazzzzah";
let haRegex = /Haz{4,}ah/; // Altere esta linha
let result = haRegex.test(haStr);
```

Especificar o número exato de capturas

Você pode especificar um número mínimo e um máximo de capturas com chaves. Às vezes, você só quer um número específico de capturas.

Para especificar este número, apenas escreva-o dentro das chaves.

Por exemplo, você pode escrever a regex `/ha{3}h/` para capturar a letra `a` `3` vezes na string `hah`.

```
let A4 = "haaaah";
let A3 = "haaah";
let A100 = "h" + "a".repeat(100) + "h";
let multipleHA = /ha{3}h/;
multipleHA.test(A4);
multipleHA.test(A3);
multipleHA.test(A100);
```

As três chamadas a `test` acima retornam, na ordem, os valores: `false`, `true` e `false`.

Exercicio = Modifique a regex `timRegex` para que capture quatro `m`s na string `Timber`.

```
let timStr = "Timmmmber";
let timRegex = /Tim{4}ber/g; // Altere esta linha
let result = timRegex.test(timStr);
```

Verificar existência

Haverá vezes em que você procurará padrões que podem ou não existir na string. Pode ser relevante validá-los dependendo da situação.

Você pode fazer com que um padrão seja opcional ao usar uma interrogação, `?`, depois dele. Ela valida se há uma ou nenhuma ocorrência do padrão. Pode-se dizer que a interrogação torna o elemento à esquerda dela opcional.

Por exemplo, com a interrogação você pode capturar palavras em inglês escritas com a ortografia americana ou britânica.

```
let american = "color";
let british = "colour";
let rainbowRegex= /colou?r/;
rainbowRegex.test(american);
rainbowRegex.test(british);
```

Ambas as chamadas ao método `test` retornam `true`.

Exercicio = Altere a regex `favRegex` para encontrar as versões americana (`favorite`) e britânica (`favourite`) da palavra.

```
let favWord = "favorite";

let favRegex = /favou?rite/; // Altere esta linha

let result = favRegex.test(favWord);
```

Usar lookaheads positivos e negativos

Lookaheads ("olhar à frente") são padrões que dizem ao JavaScript para procurar outros padrões ao longo da string sem capturá-los. Eles podem ser úteis quando é necessário fazer diversas verificações na mesma string.

Existem dois tipos de lookahead: o *lookahead positivo* e o *lookahead negativo*.

Lookaheads positivos garantem que o padrão especificado se encontra à frente, mas não o capturam. Usa-se `(?=...)`, onde `...` é o padrão a ser procurado, para escrever lookaheads positivos.

Lookaheads negativos, por outro lado, garantem que o padrão especificado não se encontra à sua frente na string. Para usar lookaheads negativos, escrevemos `(?!...)` onde `...` é o padrão que você quer ter certeza que não está lá. O restante do padrão é validado se o padrão do lookahead negativo estiver ausente.

É fácil se confundir com lookaheads, mas uns exemplos podem ajudar.

```
let quit = "qu";
let noquit = "qt";
let quRegex= /q(?=u)/;
let qRegex = /q(?!u)/;
quit.match(quRegex);
noquit.match(qRegex);
```

As duas chamadas a `match` retornam `["q"]`.

Validar dois padrões diferentes em uma string é considerado um uso mais prático de lookaheads. Neste não tão aprimorado validador de senhas, os lookaheads procuram por 3 a 6 caracteres e pelo menos um número, respectivamente, na string:

```
let password = "abc123";
let checkPass = /(?=\w{3,6})(?=\D*\d)/;
checkPass.test(password);
```

Exercicio = Use os lookaheads na `pwRegex` para que correspondam a senhas de mais de 5 caracteres e que tenham dois algarismos consecutivos.

```
let sampleWord = "astronaut";

let pwRegex =  /(?=\w{6})(?=\w*\d{2})/;// Altere esta linha

let result = pwRegex.test(sampleWord);
```

Validar grupos mistos de caracteres

Há vezes em que queremos validar grupos de caracteres em uma expressão regular. É possível fazê-lo usando parênteses: `()`.

Você pode usar a expressão regular `/P(engu|umpk)in/g` para encontrar tanto `Penguin` quanto `Pumpkin` em uma string.

Depois é só usar o método `test()` para verificar se os grupos estão presentes na string.

```
let testStr = "Pumpkin";
let testRegex = /P(engu|umpk)in/;
testRegex.test(testStr);
```

O método `test` retorna `true` aqui.

Exercicio = Corrija a regex para que ela valide os nomes `Franklin Roosevelt` e `Eleanor Roosevelt` levando em conta maiúsculas e minúsculas. A regex também deve permitir nomes do meio.

Depois corrija o código, fazendo com que a regex seja testada na string `myString`, retornando `true` ou `false`.

```
let myString = "Eleanor Roosevelt";

let myRegex = /(Franklin|Eleanor).*Roosevelt/; // Altere
esta linha

let result = myRegex.test(myString); // Altere esta linha
```

```
// Depois de passar no experimento do desafio com myString e
ver como o agrupamento funciona
```

Reusar padrões com grupos de captura

Vamos supor que você deseja encontrar a correspondência de uma palavra que ocorra várias vezes como abaixo.

```
let repeatStr = "row row row your boat";
```

Você poderia usar `/row row row/`, mas e se você não souber a palavra específica repetida? *Grupos de captura* podem ser usados para localizar substrings repetidas.

Os grupos de captura são criados envolvendo o padrão de regex a ser capturado entre parênteses. Neste caso, o objetivo é capturar uma palavra composta de caracteres alfanuméricos para que o grupo de captura seja `\w+` entre parênteses: `/(\w+)/`.

A substring correspondente ao grupo é salva em uma "variável" temporária que pode ser acessada dentro da mesma expressão regular usando uma barra invertida e o número do grupo de captura (ex.: `\1`). Os grupos de captura são automaticamente numerados pela posição de abertura de seus parênteses (esquerda para direita), começando em 1.

O exemplo abaixo captura qualquer palavra que se repita três vezes, separada por espaços:

```
let repeatRegex = /(\w+) \1 \1/;
repeatRegex.test(repeatStr); // Returns true
repeatStr.match(repeatRegex); // Returns ["row row row",
"row"]
```

Usar o método `.match()` em uma string retornará um array com a substring correspondente, juntamente com seus grupos capturados.

Exercicio = Use grupos de captura na regex `reRegex` para capturar em uma string um número que aparece exatamente três vezes, separados por espaços.

```
let repeatNum = "42 42 42";

let reRegex = /^(\d+)\s\1\s\1$/; // Altere esta linha

let result = reRegex.test(repeatNum);
```

Usar grupos de captura para buscar e substituir

Buscar texto é útil. É ainda mais útil quando você consegue modificar (ou substituir) o texto que você busca.

Você pode buscar e substituir texto em uma string usando o método `.replace()`. O primeiro parâmetro do `.replace()` é o padrão regex que você quer procurar. O segundo parâmetro é a string que substituirá o resultado da busca ou uma função que fará algo com ele.

```
let wrongText = "The sky is silver.";
let silverRegex = /silver/;
wrongText.replace(silverRegex, "blue");
```

A chamada a `replace` aqui retorna a string `The sky is blue.`.

Você também pode acessar grupos de captura na string de substituição usando o cifrão (`$`).

```
"Code Camp".replace(/(\w+)\s(\w+)/, '$2 $1');
```

A chamada a `replace` aqui retorna a string `Camp Code`.

Exercicio = Escreva uma regex, `fixRegex`, que usa três grupos de captura para procurar cada palavra na string `one two three`. Depois atualize a variável `replaceText` para trocar de `one two three` para `three two one` e

atribuir o resultado à variável `result`. Certifique-se de estar utilizando grupos de captura na string de substituição usando o cifrão (`$`).

```
let str = "one two three";
let fixRegex = /(\w+)\s(\w+)\s(\w+)/; // Change this line
let replaceText = "$3 $2 $1"; // Change this line
let result = str.replace(fixRegex, replaceText);
```

Remover espaços em branco do início e do fim de strings

Às vezes, strings têm espaços em branco indesejados em seus inícios e fins. Uma operação muito comum de strings é remover esses espaços ao redor delas.

Exercicio = Escreva uma regex que, junto dos métodos apropriados de string, remove os espaços em branco do começo e do fim delas.

Observação: normalmente, usaríamos `String.prototype.trim()` para isso, mas a sua tarefa é fazer o mesmo usando expressões regulares.

```
let hello = "   Hello, World!  ";
let wsRegex = /^\s+|\s+$/g; // Change this line
let result = hello.replace(wsRegex, ""); // Change this line
```