Programação Estruturada

Aula 07 - Operadores Javascript: Strings

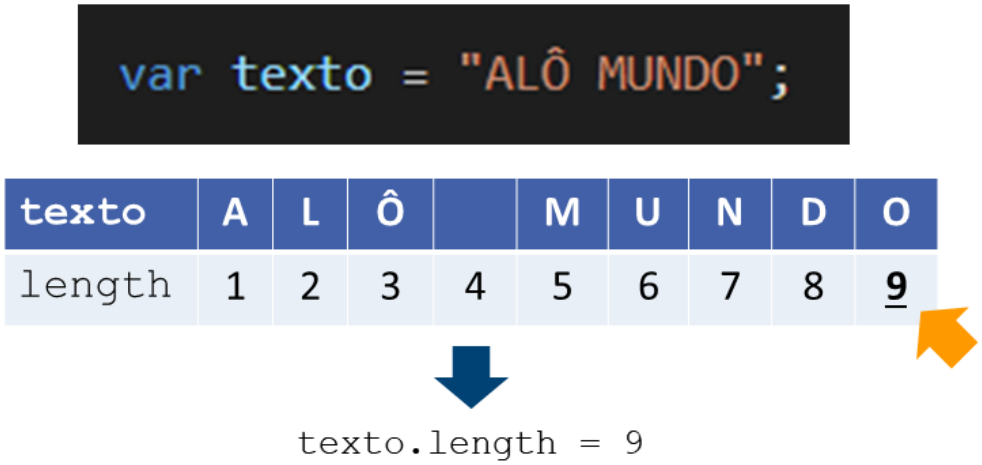
Videoaula 02 - Procurando e Extraindo *Strings*

Videoaula 02 - Procurando e Extraindo *Strings*

Como sabemos, em JavaScript, métodos e propriedades estão disponíveis para valores primitivos como, por exemplo, as *strings*. Os métodos e propriedades que JavaScript fornece para *strings* nos ajudam a trabalhar com elas.

Inicialmente, você vai conhecer a propriedade `length` das *strings*, que simplesmente retorna o comprimento de uma *string*, ou seja, sua quantidade de caracteres. Note que eu falei quantidade de caracteres e não de letras. Por exemplo, no slide, veja que o valor da expressão `texto.length` é 9. Isto porque, além das oito letras, temos também um espaço em branco, que é um caractere.

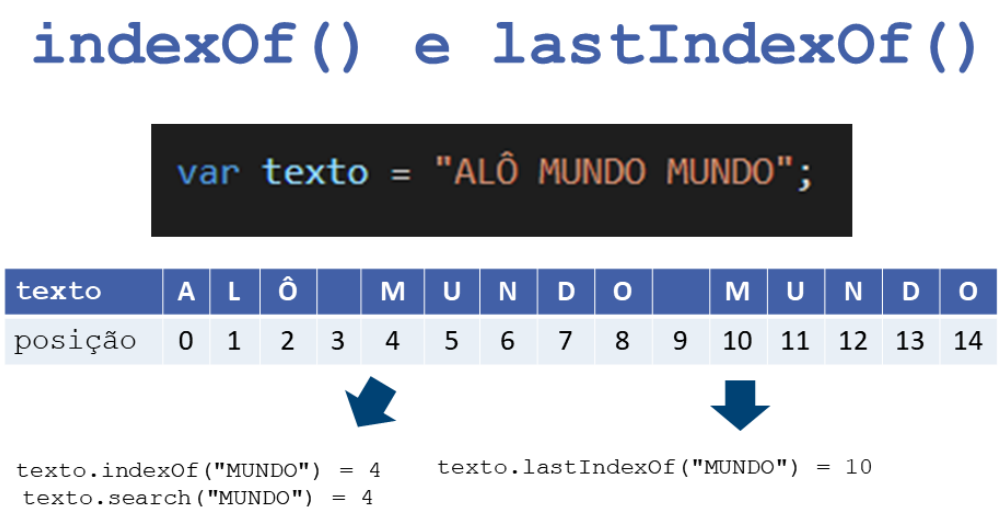
Figura 1 - Comprimento de *strings*



Muitas vezes queremos verificar se um determinado texto contém uma palavra, ou se inicia ou termina em outra etc. JavaScript nos oferece diversos métodos que nos permitem descobrir isso. O primeiro método que podemos usar em uma *string*, o `indexOf()`, recebe uma *string* como parâmetro e retorna a posição da primeira ocorrência dessa *string* na *string* em que chamamos o método. O método `search()`

também apresenta o mesmo comportamento. No slide, podemos ver que, ao chamarmos `indexOf("MUNDO")` ou `search("MUNDO")` no nosso texto, o resultado é 4.

Figura 2 - Procurando *strings*



#FicaDica

Aqui, é importante destacar o fato de que JavaScript conta posições a partir do zero. Dessa forma, a primeira posição de uma *string* é a posição 0, a segunda posição é a posição 1, e assim por diante. No slide, a posição do "A" é a posição 0 e o último "O" fica na posição 14.

Outro método, o `lastIndexOf()`, retorna o índice da última ocorrência de um texto especificado em uma *string*. Neste exemplo, ao chamarmos `lastIndexOf("MUNDO")` no nosso texto, o resultado é 10, pois a última ocorrência da *string* "MUNDO" no texto tem início nessa posição.

Uma característica importante do retorno dos três métodos, `indexOf()`, `search()` e `lastIndexOf()`, é que, se a *string* não possuir o valor passado como parâmetro, todos eles retornam -1.

A essa altura, você deve estar se perguntando por que temos dois métodos, `indexOf()` e `search()`, que são iguais. Bem, na verdade, eles não são iguais. Apenas o método `indexOf()` pode ser chamado usando um segundo parâmetro que indica a partir de que posição a busca deve começar. Além disso, apenas o método `search()` pode ser chamado usando uma expressão regular como parâmetro. Não iremos abordar expressões regulares nesta disciplina, mas de maneira bastante resumida, uma expressão regular é um objeto que descreve um padrão de caracteres e que pode ser usado para fazer o que chamamos de casamento de padrão ou para "pesquisar e substituir" no texto.

Vejamos agora alguns exemplos de utilização desses métodos de busca de *strings*.

Neste exemplo, definimos a variável `texto` e ela recebe as palavras "ALÔ MUNDO MUNDO", todas em letras maiúsculas e com um espaço em branco entre as palavras "ALÔ MUNDO" e entre as palavras "MUNDO MUNDO". Alterando o valor da variável chamada `resultado`, como mostra na linha 26 da figura 3, escrevendo o valor dela na página HTML.

Figura 3 - Exemplo de busca em *strings*

```
14
15      var texto = "ALÔ MUNDO MUNDO";
16
17      var resultado = texto.length;
18      //var resultado = texto.indexOf("MUNDO");
19      //var resultado = texto.indexOf("MUNDO", 4); // Altere 4 para 5
20      //var resultado = texto.search("MUNDO");
21      //var resultado = texto.lastIndexOf("MUNDO");
22
23      //var patt = /[uo]/i;
24      //var resultado = texto.search(patt);
25
26      document.getElementById("saida").innerHTML = resultado;
27      </script>
28  </body>
```

A primeira atribuição que vou fazer está na linha 17, é do valor de `texto.length`, ou seja, do comprimento da variável `texto` que é uma *string*. Na tela HTML, dar para ver o comprimento 15, isso porque nós temos 3 da palavra "ALÔ", com o espaço em branco fica 4, mais 5 da palavra "MUNDO", então nós temos 4 + 5 = 9, e com outro espaço em branco, 10. Acrescentamos as cinco letras da palavra "MUNDO", assim, de fato, o comprimento dessa *string* "ALÔ MUNDO MUNDO" é 15, deu para entender? Então, sempre que eu quiser saber o comprimento de uma *string*, posso usar o `.length` para isso.

Outro teste que vou fazer é perguntar qual é o índice do texto "MUNDO". Para isso, aplico o método `indexOf("MUNDO")` à variável `texto`, na linha 17, e o resultado que nós temos é o 4, sabe o porquê? O primeiro índice é o 0, então temos o "A" sendo o índice 0, o "L" no índice 1, o "Ô" no índice 2, o espaço em branco no índice 3, em seguida, temos o começo da primeira ocorrência da palavra "MUNDO" no índice 4, então por isso que a resposta é, como podemos ver na página do HTML, o 4.

Mais um teste é perguntar qual é o índice usando o método `indexOf()` do texto "MUNDO", mas começando a partir do caracter 4, ou seja, da posição 4 do texto. Após recarregar a página HTML, continuamos tendo a mesma resposta que é o 4 porque, como vimos no último teste, a primeira ocorrência da palavra "MUNDO" começa, de fato, na posição 4. Fazendo outro teste, alterando na linha 17, do 4 pra 5, como mostra o trecho do código a seguir.

Código 1 - 07_4 Procurando Strings.html

```
1  <html>
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 07</title>
5    </head>
6    <body>
7      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
8
9      <h1>Procurando Strings</h1>
10
11     <p id="saida"></p>
12
13     <script>
14
```

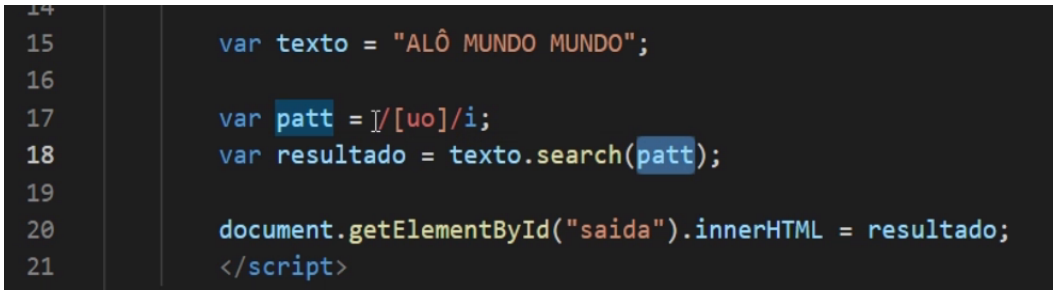
```
15      var texto = "ALÔ MUNDO MUNDO";
16
17      var resultado = texto.indexOf("MUNDO", 5); // Altere 4 para 5
18      //var resultado = texto.search("MUNDO");
19      //var resultado = texto.lastIndexOf("MUNDO");
20
21      //var patt = /[uo]/i;
22      //var resultado = texto.search(patt);
23
24      document.getElementById("saida").innerHTML = resultado;
25
26      </script>
27    </body>
28 </html>
29
```

Ao recarregar a página HTML, alterou para a posição 10, isso porque se olhar no texto, dar para ver o 4 até a palavra "MUNDO", mais 5 da própria palavra "MUNDO",temos 9. E a segunda ocorrência da palavra "MUNDO" começa, de fato, na posição 10, certo? Então, desconsideramos o "ALÔ M", ou seja, é como se tivesse procurando no texto "UNDO MUNDO". Então, começamos a procurar o texto "MUNDO" a partir da letra "U" da primeira palavra "MUNDO", na linha 15, como pode ser visto na imagem anterior.

No próximo teste, vai ser usado o `search()`, passando a palavra "MUNDO" e o resultado é o 4. Sabe por quê? Porque começa no índice 4 a palavra "MUNDO". Agora, se eu usar o `lastIndexOf("MUNDO")`, ele vai dar o índice de quando começa a última ocorrência da palavra "MUNDO". Como percebemos, a segunda ocorrência da palavra "MUNDO" inicia no indíce 10, que é exatamente o que fica escrito na página HTML quando a gente procura o `lastIndexOf()` da palavra "MUNDO" nesse texto.

Nesta videoaula, foi mencionada a existência de padrões e que é possível procurá-los usando o `search()`. Vamos localizar o padrão que está definido na linha 17, como mostra a figura 4, e, para isso, passamos para o método `search()` no campo texto e obtemos o índice 5 na página HTML.

Figura 4 - Procurando padrões em *strings*



Afinal de contas, que padrão é esse? O que estamos procurando são as letras "u" ou "o" e não considerando se é maiúscula ou minúscula. Logo, qualquer coisa que for "u" ou "o", independente de ser minúscula ou maiúscula, vai se encaixar com esse padrão. Então, vejam na linha 15, que partindo do índice 0, quando chegamos no índice 5, encontramos a letra "U", que, independentemente de ser maiúscula, como foi definida no padrão, a resposta será 5.

Como não iremos nos aprofundar em padrões, pois isso não será abordado nesta disciplina, vocês podem ver em outro momento.

Saiba Mais

Caso você deseje saber mais sobre padrões, pode acessar um desses links:

- JavaScript RegExp Reference
- https://www.ecma-international.org/ecma-262/9.0/index.html#sec-regexp-regular-expression-objects

Outra tarefa comum quando trabalhamos com *strings* é a extração de partes delas. Para isso, três métodos bastante usados em JavaScript são `slice()`, `subString()` e `substr()`.

O método `slice()` recebe dois parâmetros, a posição inicial e a posição final, e retorna uma nova *string* resultante de extrair a parte da *string* que começa na posição inicial e termina antes da posição final. Na figura 6, vemos que, ao chamarmos `slice(4,11)`, no nosso texto temos como resultado o texto "MUNDO L". Isso porque esse é o texto que temos da posição 4 à posição 11 – 1, ou seja, a posição 10.

Figura 5 - Extraindo *strings*

Aqui, é muito importante reforçar o fato que o método `slice()` não altera a *string* original `texto`, mas retorna uma nova *string* como resultado. Essa é uma característica de todos os métodos de *string*, incluindo `subString()` e `substr()`. Isso porque, em JavaScript *strings* são imutáveis, ou seja, elas não podem ser alteradas, apenas substituídas.

Se um parâmetro for negativo, a posição será contada a partir do final da *string*. Na imagem abaixo, vemos que, ao chamarmos `slice(-11,-4)` no nosso texto, temos como resultado o mesmo texto, ou seja, "MUNDO L". Isso porque esse é o texto que temos, contando de trás para frente, da posição -11 à posição -4 – 1, ou seja, -5.

Figura 6 - Extraindo *strings*

Esse método também pode ser usado com apenas um parâmetro. Nesse caso, ele retorna uma nova *string* resultante de extrair a parte da *string* que começa na posição passada. No slide, vemos que, ao chamarmos `slice(4)` no nosso texto, temos como resultado o texto "MUNDO LINDO". Isso porque esse é o texto que temos a partir da posição 4.

Novamente, se esse único parâmetro for negativo, a posição será contada a partir do final da *string*. No slide, vemos que, ao chamarmos `slice(-11)` no nosso texto, temos como resultado o mesmo texto, "MUNDO LINDO". Isso porque esse é o texto que temos a partir da posição -11, contando a partir do final.

Atenção

É importante saber que posições negativas não funcionam no Internet Explorer 8 e nas versões anteriores.

Agora, vamos falar sobre o método `subString()`. Ele é praticamente idêntico ao método `slice()`. A única diferença é que parâmetros negativos são interpretados como 0. No slide, vemos que ao chamarmos `subString(4,11)` no nosso texto temos também como resultado o texto "MUNDO L".

Além disso, da mesma maneira que o método `slice()`, ao chamarmos `subString(4)` no nosso texto, temos como resultado o texto "MUNDO LINDO".

Por fim, podemos também extrair parte de uma *string* usando o método `substr()`, o qual também é muito similar ao `slice()`. A diferença é que o segundo parâmetro especifica o comprimento da parte extraída. Além disso, assim como no método `subString()`, parâmetros negativos são interpretados como 0.

Por exemplo, no slide vemos que, ao chamarmos `substr(4,7)` no nosso texto, temos como resultado o texto "MUNDO L". Isso porque se extrairmos sete caracteres a partir da posição 4, ou seja, "M" (1), "U" (2), "N" (3), "D" (4), "O" (5), o espaço em branco (6) e "L" (7), obtemos essa *string*.

Se você omitir o segundo parâmetro, `substr()`, cortará o restante da *string*. Por isso, da mesma maneira que o método `slice()` e o método `subString()`, ao chamarmos `substr(4)` no nosso texto, temos como resultado o texto "MUNDO LINDO".

Caso você deseje retornar apenas o caracter em uma determinada posição, pode utilizar o método `charAt()`. No slide, como podemos ver abaixo, ao chamarmos `charAt(4)` no nosso texto, temos como resultado o texto "M", ou seja, o texto contendo apenas o caractere que está na posição 4.

Figura 7 - Retornando caracteres

#FicaDica

A partir da versão ECMAScript 5, de 2009, JavaScript permitiu fazer o mesmo utilizando *strings* como arrays, como podemos ver no slide. No entanto, o resultado da expressão `texto[4]` é um pouco imprevisível, visto que ela não funciona no Internet Explorer 7 ou em versões anteriores. Além disso, temos um problema conceitual, pois essa expressão faz com que *strings* pareçam ser arrays, o que não é verdade. Por isso, recomendamos sempre acessar um caractere de uma *string* usando o método `charAt()`.

No entanto, algumas vezes desejamos, de fato, separar os elementos de uma *string* usando um caractere separador. Para isso, podemos usar o método `split()`. Porém, vamos deixar para conhecer melhor esse método, ainda nesta disciplina, na aula que teremos sobre arrays.

Agora que você já sabe como funcionam os métodos `slice()`, `subString()`, `substr()` e `charAt()`, pode utilizar o arquivo **07_5_Extraindo_Strings.html** para verificar o comportamento deles na prática. Fique à vontade para editar esse arquivo a fim de fazer vários testes.

Código 2 - 07_5 Extraindo Strings.html

```
1  <html>
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 07</title>
5    </head>
6    <body>
7      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
8
9      <h1>Extraindo Strings</h1>
10
11     <p id="saida"></p>
12
13     <script>
```

```
14
15      var texto = "ALÔ MUNDO LINDO";
16
17      var resultado = texto.slice(4,11);
18      //var resultado = texto.slice(-11, -4);
19      //var resultado = texto.slice(4);
20      //var resultado = texto.slice(-11);
21
22      //var resultado = texto.substring(4,11);
23      //var resultado = texto.substring(4);
24      //var resultado = texto.substring(-1, -10);
25
26      //var resultado = texto.substr(4,7);
27      //var resultado = texto.substr(4);
28      //var resultado = texto.substr(-1, -10);
29
30      //var resultado = texto.charAt(4);
31      var resultado = texto.charAt(100);
32
33      document.getElementById("saida").innerHTML = resultado;
34
35      </script>
36    </body>
37  </html>
38
```

Até a próxima videoaula!