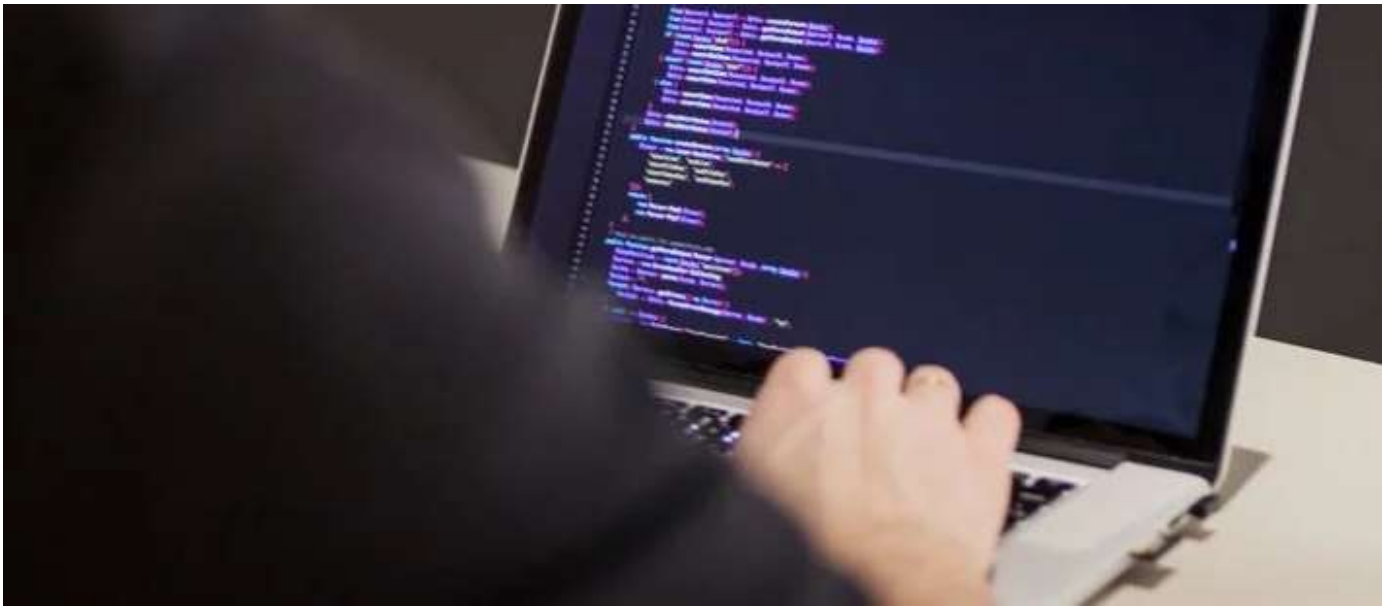
Strings com JavaScript: o que são e como manipulá-las



André Bessa
Atualizado em 13/09/2021

Em programação, a todo momento trabalhamos com **dados dos mais variados tipos**, independentemente da linguagem.

Alguns exemplos são os booleanos (verdadeiro ou falso), números (inteiros, ponto flutuante) e alguns tipos mais complexos, como estruturas e objetos. **Um dos tipos mais utilizados em programação são as *strings***, sequências (ou cadeias) de caracteres que usamos para, entre outras coisas, manipular textos.

No exemplo a seguir, temos uma cadeia de caracteres representada como um array:

```
const fruta = "banana"
// ["b", "a", "n", "a", "n", "a"]
```

Imagine que estamos desenvolvendo o código de uma aplicação em JavaScript. A validação dos campos de login e senha foi solicitada e é preciso verificar se o tamanho da senha de cada usuário atende à regra de ter um tamanho mínimo de 8 caracteres e impedir que haja espaços no início ou no fim do login cadastrado.

Desse modo, precisaremos fazer o tratamento de strings.

Como podemos fazer isso? Quais opções o JavaScript pode oferecer para esses casos? No texto a seguir, vamos ver essas possibilidades de trabalhar com strings na linguagem.

O que é uma String?

Por definição, strings são sequências de caracteres alfanuméricos (letras, números e/ou símbolos) amplamente usadas em programação. Em Javascript, uma string sempre estará entre aspas.

```
const frase = "Mergulhando em tecnologia com Alura";
```

ou

```
const frase = ‘Mergulhando em tecnologia com Alura’;
```

ou ainda

```
console.log('Mergulhando em tecnologia com Alura')
```

Espere! Mas eu declaro minhas strings com aspas duplas ou simples?

Podemos colocar nossas strings entre aspas duplas ou simples. Para o JavaScript, não há diferença, já que ele considera as duas formas de declaração válidas. Mas, atenção, essa regra pode não se aplicar a outras linguagens. No Java ou C#, por exemplo, aspas simples são usadas para definir um caractere.

Em alguns momentos, a string poderá ser um texto que contém aspas. Nesses casos, é preciso combinar a utilização das aspas simples com aspas duplas e vice-versa, porque um texto como: `“Ela disse: “Adeus””` , não funciona corretamente.

Vamos ao exemplo:

```
console.log('Ela disse: “Adeus!”')
```

ou

```
console.log(“Ela disse: 'Adeus!' ”)
```

É importante ressaltar que, depois que a sequência de caracteres for definida, a string é imutável, ou seja, não poderá ter seu valor alterado. Então, como manipular a string?

Sempre que manipulamos uma string, é criada uma nova instância dela por baixo dos panos, o que significa que será gerado um novo espaço na memória com uma cópia do valor da string. Por isso, temos que utilizar uma variável para armazená-la.

Objeto String

A linguagem JavaScript traz ainda como recurso um objeto global `String` que nos permite criar ou converter um tipo em uma string, veja o exemplo abaixo:

```
const numero = 256
const convertidoEmString = new String(numero)
```

A saída após exibirmos a variável `convertidoEmString` usando o método `console.log()` é `[String: '256']`. Na construção do objeto usando `new String(parâmetro)`, o parâmetro pode ser qualquer elemento do nosso código que queiramos transformar em string.

Também é possível converter outros tipos primitivos (por exemplo, números e booleanos) em strings com o método `toString()`:

```
const num = 500
console.log(num.toString()) //'500'
```

Usando Strings

É possível interpolar, concatenar, checar posições de caracteres ou ainda substituir partes de strings. Vamos ver algumas dessas utilizações com o JavaScript?

Concatenando strings

Quando falamos em concatenar strings, quer dizer que vamos juntar duas ou mais strings e formar uma nova. Observe o exemplo abaixo:

```
let nome = "André"
let sobreNome ="Silva"
let nomeCompleto = "Meu nome completo é : " + nome + sobreNome
```

Para concatenar as strings `nome` e `sobreNome` com a string de texto que é o valor de `nomeCompleto`, usamos o operador de adição (`+`). Podemos usar também `+=`, como no exemplo abaixo:

```
let nome = "André"
let saudacoes = "Seja bem-vindo "
saudacoes += nome
```

Dessa forma, temos a saída `Seja bem-vindo André`

Interpolando strings (template strings)

A interpolação de strings é um recurso bem interessante, presente em diversas linguagens. No JavaScript, é uma alternativa mais prática para manipular string sem a necessidade de fazer concatenação, porque para textos maiores, concatenar pode ser um pouco trabalhoso.

Usando as chamadas *template strings* ou templates literais, a pessoa desenvolvedora consegue ter uma flexibilidade maior no trabalho com strings, além de facilitar a escrita e leitura do código.

Retomando o exemplo da mensagem de boas vindas, veja abaixo a utilização de *template strings*:

```
let nome = "André"
let saudacoes =`Seja bem-vindo ${nome}`
```

Veja como exemplo o poema “E agora, José?” de Carlos Drummond de Andrade:

```
let nome = "André"
let poema = `
   E agora, ${nome}?
   A festa acabou,
   a luz apagou,
   o povo sumiu,
   a noite esfriou,
   e agora, ${nome}?
   e agora, você?
   você que é sem nome,
   que zomba dos outros,
   você que faz versos,
   que ama, protesta?
   e agora, ${nome}?
`
```

Observe que, para a utilização da *template string*, ela deve estar entre acentos graves (`) e, para fazer a interpolação, o valor ou variável deve estar dentro da estrutura ${valor} . Vale ressaltar que usando *template strings* temos a opção de utilizar a quebra de linha normalmente, sem caracteres de escape para isso, como \n`.

Métodos para strings

Antes de começarmos, é importante ressaltar que o JavaScript diferencia strings como tipos primitivos (com aspas duplas ou simples) de objetos Strings (quando usamos a palavra reservada `new`). Mas, por baixo dos panos toda string, mesmo as que criamos com a chamada “forma literal”, por exemplo `const texto = “Alura”` , acaba convertida para

um objeto do tipo `String`. Por isso, temos acesso a uma série de métodos e propriedades deste objeto.

Agora que entendemos isso, vamos ver algumas propriedades e métodos úteis e bem práticos para trabalhar com strings em nossas aplicações.

`.length`

A propriedade `length` serve para nos informar o tamanho de uma string. E por que isso é útil?

Caso sua aplicação tenha como uma das regras para criação de senhas (que em geral são alfanuméricas) o tamanho de 8 caracteres, usar `length` será uma boa opção, pois ajudará a contar a quantidade de caracteres da string.

Para testar a propriedade `length`, vamos usar a string `alura`, que retornará o tamanho 5.

```
const palavra="alura";
console.log(palavra.length) //5
```

Veja que `length` é exatamente a mesma propriedade que acessamos quando queremos descobrir o comprimento (ou seja, a quantidade de elementos) em um array.

`charAt()`

Com o método `charAt()` conseguimos acessar um caractere de uma string. Lembre-se que, por baixo dos panos, strings são arrays de caracteres, e em cada posição temos o caractere que compõe a string.

Veja o exemplo abaixo:

```
console.log("alura".charAt(3)) //r
```

Após a execução do método `charAt()`, ela retornará o caractere `r`, que é o valor que consta na posição 3 da string - lembrando que arrays em JavaScript começam na posição 0 (zero).

Outra alternativa será usar a notação de colchetes para encontrar um caractere da string, da seguinte forma:

```
const palavra="Alura"
console.log(palavra[0]) //A
```

Será exibido o caractere `A`, ou seja, o que está na primeira posição da string. O resultado da execução do `charAt()` é uma string.

Mas e se quisermos saber qual a posição de um caractere dentro da string?

`indexOf()`

Respondendo a pergunta anterior, existe a função `indexOf()`, que retorna a posição de um caractere dentro da string.

Por exemplo:

```
const palavra="Alura"
console.log(palavra.indexOf("a")) //4
```

O resultado é a posição 4. Porém, na utilização do `indexOf()`, fique atento caso o caractere que se busca na string seja encontrado em mais de uma posição, pois será retornada somente a primeira ocorrência. veja o código abaixo:

```
const palavra="Divertidamente"
console.log(palavra.indexOf("e")) //3
```

O resultado da execução do `indexOf()` é um valor numérico.

`toUpperCase()` e `toLowerCase()`

São duas funções bastante utilizadas quando estamos trabalhando com string e precisamos deixar o texto todo em letras minúsculas (*lower case*) ou todo em maiúsculas (*upper case*). Vamos ver o código abaixo:

```
const palavra="alura";
console.log(palavra.toUpperCase()) //ALURA
console.log(palavra.toLowerCase()) //alura
```

Após a execução do código, o console irá exibir `ALURA` e `alura` respectivamente. O resultado da execução dos métodos `toUpperCase()` e `toLowerCase()` é uma nova string .

`substr()`

Outra função muito interessante é a `substr()` (*substring*), que permite que façamos a extração de parte de uma string, conforme o código abaixo:

```
let frase= "Mergulhando em tecnologia."
console.log(frase.substr(0,11)) // Mergulhando
```

A função recebe como parâmetro o início e o fim da nova string a ser retirada da string principal. Na execução do código acima, temos como resultado a palavra `Mergulhando` . Bem útil, né?

O resultado da execução do método `substr()` é uma nova string .

`slice()`

Podemos utilizar também o método `slice()` , que usamos com arrays. Ele é similar ao `substring()` e retornará parte de uma string, desde que passemos nos parâmetros o índice de início e de fim. Veja abaixo:

```
let frase= "Mergulhando em tecnologia."
console.log(frase.slice(0,11)) // Mergulhando
```

O resultado da execução do método `slice()` é uma nova string .

`replace()`

Com a função `replace()` temos a possibilidade de substituir parte de uma string por outra. Essa função recebe como parâmetros duas informações: a string que você quer

substituir e a string que será colocada no lugar. Olhe o exemplo abaixo, em que precisamos substituir a string `nomeusuario` no texto padrão de `comunicacao`.

```
let nome = "André";
let comunicacao = " Olá, sr. nomeusurario, informamos que a partir da pre
console.log(comunicacao.replace("nomeusurario",nome));
```

Na execução deste exemplo, a string `nomeusuario` será substituída pelo conteúdo da variável `nome`. Como resultado da execução do método `replace()` teremos uma nova string.

`concat()`

O método `concat()` é uma opção para concatenar strings sem a utilização do operador de adição (`+`). Ele concatena duas strings, adicionando a nova string ao fim da anterior.

Observe uma utilização do `concat()`:

```
= "Programe nas principais linguagens e plataformas. Explore linguagens co
vaString.concat("JavaScript,").concat(" Python,").concat(" e C#."))
```

O resultado obtido será: `Programe nas principais linguagens e plataformas. Explore linguagens como [JavaScript](https://www.alura.com.br/artigos/javascript), [Python](https://www.alura.com.br/artigos/python), e C#.`

Para a execução do método `replace()` teremos como resultado uma nova string.

`split()`

O método `split()` é bem interessante, pois com ele conseguimos quebrar uma string com base em caracteres separadores que vamos informar para o método como parâmetro.

Vamos ver um exemplo:

```
let linguagens = "JavaScript;Java;C#;PHP;Python;Go;Vb;SQL;C;C++";
let arrayLinguagens = linguagens.split(";");
console.log(arrayLinguagens)
```

Quando trabalhamos com o `split()`, devemos nos atentar, pois a execução gerará como resultado um array de strings com os elementos que foram separados com base no separador desejado. Portanto a execução do código resulta em um array como mostrado a seguir:

```
[ 'JavaScript',
  'Java',
  'C#',
  'PHP',
  'Python',
  'Go',
  'Vb',
  'SQL',
  'C',
  'C++' ]
```

Lembre-se que o resultado da execução do método `split()` é um array de strings.

`trim()`

O `trim()` remove os espaços em branco **no início ou fim** de uma string. Se em alguma situação precisarmos fazer uma verificação de que o usuário não digitou o login com espaços, faremos;

```
let login = "   andre@emailteste.com      ";
let loginSemEspaco = login.trim();
console.log(loginSemEspaco); //andre@emailteste.com
```

A variável `loginSemEspaco` conterá uma nova string, sem os espaços em branco no início ou fim que podem ter sido digitados. Então, quando executado o método `trim()`, o

resultado é uma nova string.

No JavaScript ainda temos algumas variações desta função como: `trimEnd()`, `trimStart()`, `trimLeft()` e `trimRight()`, teste também estas variantes e veja o resultado obtido, ok?

Conclusão

Neste artigo, vimos o que são strings e como podemos manipulá-las usando métodos do Javascript.

Lembre-se que trabalhar com texto é uma atividade que todas as pessoas que desenvolvem farão em seus códigos em algum momento. Por isso é tão importante conhecer as strings e suas particularidades. E aí, curtiu? Vamos programar!

Para saber mais sobre manipulação de strings, veja:

- Curso Avançando com PHP: Arrays, Strings, Função e Web na Alura
- Curso Fundamentos do JavaScript: Tipos, variáveis e funções na Alura
- JavaScript replace: manipulando Strings e regex

Confira neste artigo:

André Bessa

Eu sou programador e instrutor de programação usando C# e .NET. Formado em Sistemas de Informação. já programei usando Java, PHP,C#, PostgreSQL e MySQL, além de já ter atuado com suporte também. Buscando sempre aprender mais sobre tecnologias.Hobbies são gibis e séries.

Artigo Anterior

Algoritmos e Lógica de programação: O que são e qual a importância?

Próximo Artigo

O que são as tipagens estática e dinâmica em programação

Leia também:

- Guia de JavaScript: o que é e como aprender a linguagem mais popular do mundo?
- Formatando e Arredondando números no JavaScript: 2 casas decimais e outros casos possíveis
- Async/await no JavaScript: o que é e quando usar a programação assíncrona?
- JavaScript: convertendo String para número
- Criando uma máscara de Telefone com Javascript
- Javascript ou Typescript?
- HTML, CSS e Javascript, quais as diferenças?

Veja outros artigos sobre Programação

Artigos

Podcasts

Artigos de educação corporativa

Novidades e Lançamentos

Email* ENVIAR

CURSOS

Cursos de Programação
Lógica | Python | PHP | Java | .NET | Node JS | C | Computação | Jogos | IoT

Cursos de Front-end
HTML, CSS | React | Angular | JavaScript | jQuery

Cursos de Data Science
Ciência de dados | BI | SQL e Banco de Dados | Excel | Machine Learning | NoSQL | Estatística

Cursos de Inteligência Artificial
IA para Programação | IA para Dados

Cursos de DevOps
AWS | Azure | Docker | Segurança | IaC | Linux

Cursos de UX & Design
Usabilidade e UX | Vídeo e Motion | 3D

Cursos de Mobile
React Native | Flutter | iOS e Swift | Android, Kotlin | Jogos

Cursos de Inovação & Gestão