

Linguagem de programação: Javascript

Jailson Costa dos Santos



Por que aprender Javascript?

Além dele ser o último das 3 linguagens ligadas ao front-end para serem aprendidas. Ele é responsável por dispor interatividade à sua página, ademais, também é possível fazer diversas coisas com ele.
Como:

- ▶ Com ele é possível mudar textos HTML, valores e seus código.
- ▶ Consegue mudar os valores de propriedades em CSS.
- ▶ É orientado em eventos e objetos, podendo ser ativado quando realizar determinada ação, como apertar um botão, etc.

Métodos de utilização do Javascript no seu código

O Javascript possui 2 formas de ser utilizado pelo usuário para escrever seu código, sendo os métodos interno e externo.

- ▶ Método interno = Se escreve a tag <script> no **final** do header **ou** do body. Por exemplo:

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

OBS: Porém, não é recomendado colocá-lo no header, uma vez que fará a interatividade do site ser carregada antes de suas funções.

- ▶ Método externo = Se escreve a tag <script> no final do body, porém adiciona o atributo 'src' nela, colocando o caminho do arquivo que possuir a extensão **.js**. Por exemplo:

```
<script src="myScript.js"></script>
```

Vantagens de se usar o Javascript Externo:

- ▶ Quando uma URL é colocada no **src**, faz com que a página pese menos no servidor, logo, sendo carregada mais rápido.
- ▶ Permite separar código HTML e Javascript, facilitando leituras posteriores (depois de um tempo).
- ▶ Pode encaminhar vários códigos Javascript em apenas 1 página HTML, por exemplo:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

Formas de saída no Javascript

É possível escrever um texto no Javascript e fazê-lo aparecer direto na sua página. Tendo 4 formas de se obter este resultado, sendo elas:

- ▶ innerHTML = Permite trocar o texto HTML, pode ser colocado usando um parágrafo e o chamando através de um ID. Por exemplo:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
```

- ▶ document.write() = É exibido diretamente na página, é como se ele fosse adicionar um novo parágrafo com o conteúdo informado.

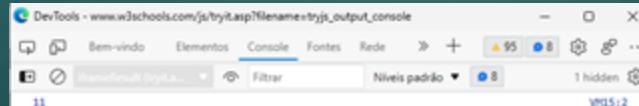
```
<script>
document.write(5 + 6);
</script>
```

Formas de saída no Javascript

- ▶ `window.alert()` = É criado um pop-up assim que a página carregar, ou caso seja realizado um determinado evento. Por exemplo:



- ▶ `alert()` = No caso ele é uma versão abreviada do `window.alert()`
- ▶ `console.log()` = Escreve no console da página, normalmente é usado para teste de determinados eventos.



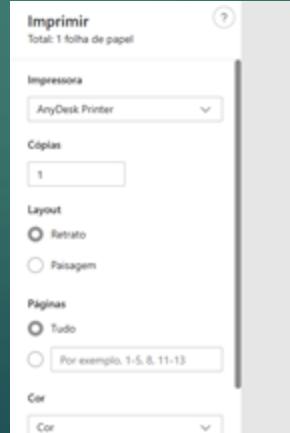
Impressão de página no Javascript

Caso você resolva trabalhar de jornalista e escreva a matéria na sua página da internet. Saiba que também é possível imprimi-la sem muitos problemas, apenas escrevendo o método ‘window.print()’, não se preocupem que verão mais a respeito de métodos no futuro. Mas o resultado deste código será este:

O método window.print()

Clique no botão para imprimir esta página.

[Imprimir página](#)



06/03/2023, 18:01

WSLUbuntu Try Editor

O método window.print()

Clique no botão para imprimir esta página.

[Imprimir página](#)

Instruções no Javascript

Um programa javascript é composto por uma lista de instruções passadas via código javascript que serão executadas.

As instruções são compostas de: Valores, Operadores, Palavras-chave e Comentários.

Por exemplo: Esta declaração diz ao navegador para escrever “Hello Dolly” em um elemento HTML que possua o ID “demo”:

```
document.getElementById("demo").innerHTML = "Hello Dolly.;"
```

A ordem das instruções serão executadas de cima para baixo e da esquerda para direita.

Instruções no Javascript

O ponto e vírgula (;) separam instruções no javascript, permitindo que você escreva diversas delas em uma única linha. Por exemplo:

```
a = 5; b = 6; c = a + b;
```

```
let a, b, c; // Declarando 3 variáveis;  
a = 5;        // Definindo o valor de a;  
b = 6;        // Definindo o valor de b  
c = a + b;    // Definindo que o valor de C é a soma de a + b
```

Em javascript, o ponto e vírgula (;) só é obrigatório caso seja escrito mais de uma instrução na mesma linha, porém é sempre aconselhável manter o ritmo de colocá-lo após terminar uma linha de código.

Instruções no Javascript

O javascript ignora diversos espaços em branco, de certa forma que esses códigos terão o mesmo resultado:

```
let person = "Hege";
let person="Hege";
```

Porém, é uma boa prática sempre adicionar espaços entre operadores (+ , - , * , / , =). Por exemplo:

```
let x = y + z;
```

Instruções no Javascript

Para melhor legibilidade, os programadores gostam de evitar linhas de código com mais de 80 caracteres, e por isso acabam as quebrando e passando para outra linha. O melhor lugar para fazer uma quebra de linha é depois de um operador. Por exemplo:

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

Instruções no Javascript

Estas instruções também podem ser colocadas dentro de blocos de código (entre chaves{}) para quando chamadas, serem executadas em conjunto. O exemplo mais claro a respeito disso são as funções, como:

```
<script>
function myFunction() {
    document.getElementById("demo1").innerHTML = "Ta fácil?";
    document.getElementById("demo2").innerHTML = "Jailson a caminho";
}
</script>
```

Clique aqui se ta fácil!

Clique aqui se ta fácil!

Ta fácil?

Jailson a caminho

OBS: Não se preocupe que verá mais sobre funções lá na frente.

Palavras-Chave no Javascript

Vou mostrar uma lista de palavras-chave (realizam determinada ação dentro do javascript) que não podem ser escolhidas como nome de variáveis.

Palavra-chave	Descrição
var	Declara uma variável
let	Declara uma variável de bloco
const	Declara uma constante de bloco
if	Marca um bloco de instruções a serem executadas em uma condição
switch	Marca um bloco de instruções a serem executadas em diferentes casos
for	Marca um bloco de instruções a serem executadas em um loop
function	Declara uma função
return	Sai de uma função
try	Implementa o tratamento de erros em um bloco de instruções

Sintaxe no Javascript

Os slides de sintaxe serão para explicar como é escrito determinadas linhas de código, é necessário para evitar possíveis erros de interpretação do navegador.

Sintaxe no Javascript: valores

Existem 2 tipos de valores no javascript, sendo eles:

- ▶ Literais = Valores que não são armazenados em variáveis.
- ▶ Variáveis = Valores que são armazenados em variáveis.

Os números nos valores literais podem ser escritos com ou sem casas decimais. Assim como textos podem ser escritos entre aspas duplas ou simples. Por exemplo:

10.50

1001

"John Doe"

'John Doe'

Sintaxe no Javascript: variáveis

As variáveis podem ser declaradas através das palavras-chave 'var' ou 'let'. Após escritas, será informado o nome da variável e o valor que ela irá receber. Por exemplo:

```
let x = 6;
```

As variáveis também podem ser declaradas e receberem seus valores em linhas diferentes. Por exemplo:

```
let x;  
x = 6;
```

OBS: A palavra-chave 'var' caiu em desuso. Não é recomendável usá-la.

Sintaxe no Javascript: variáveis

As variáveis possuem regras para quando forem declaradas, sendo elas:

- ▶ Deve começar com uma letra, maiúscula ou minúscula.
- ▶ Pode começar com um cifrão (\$).
- ▶ Pode começar com um sublinhado (_).
- ▶ Não pode começar com números.
- ▶ Não pode conter espaços, por exemplo: 'meu nome = "Alvaro" '.
- ▶ Não pode ser uma palavra-chave, também conhecida como palavra reservada.
- ▶ Pode conter números **após** escrever o primeiro caractere dentre os permitidos.
- ▶ Não pode conter caracteres especiais fora os ja mencionados.

Sintaxe no Javascript: operadores

O javascript utiliza operadores aritméticos para calcular determinados valores. Por exemplo (* / + -):

```
(5 + 6) * 10
```

Além disso, é utilizado o operador **igual** para **atribuir** valores (=). Por exemplo:

```
let x, y;  
x = 5;  
y = 6;
```

OBS: A partir de agora, crie o hábito de ler o operador **igual** (=) como **recebe**. No caso acima, 'x **recebe** 5'.

Sintaxe no Javascript: concatenação

O operador de adição (+), além de servir para somar os números, também serve para concatenar palavras. Por exemplo:

```
document.getElementById("demo").innerHTML = "Eu " + "Não " + "Sei";
```

Eu Não Sei

Sintaxe no Javascript: comentários

Os comentários são muito úteis para o usuário pois além de serem utilizados para guiar outras pessoas pelo seu código, ele também serve para marcar seções ou avisos para outras pessoas, uma vez que o navegador ignora comentários no código.

Para comentar apenas **uma** linha, utiliza-se duas barras (//). Podem ser utilizadas da seguinte maneira:

```
x = 5; // Vai ser executado  
// x = 6; Não vai ser executado
```

Para comentar mais de uma linha, utiliza-se /* para abrir o comentário e um */ para fechar o comentário. Por exemplo:

```
/*  
Este  
é um  
comentário  
multi-linha  
*/
```

Sintaxe no Javascript: case sensitive

O javascript diferencia letras maiúsculas de minúsculas, por exemplo:

A variável ‘nome’ é diferente da variável ‘NOME’ que é diferente da variável ‘Nome’.

A maioria das palavras-chave são definidas em letras minúsculas, caso contrário, o javascript não consegue reconhecê-las.

Variáveis no Javascript

O que são variáveis?

- ▶ Pense nelas como uma espécie de gaveta para guardar coisas, como seus apontadores, borrachas, etc.
- ▶ Dessa forma, a variável é uma gaveta que serve para guardar textos, números, listas, entre outras coisas.
- ▶ Elas **recebem** seus valores através do operador de atribuição `(=)`.

Variáveis no Javascript: palavras-chave

Existem 4 maneiras de declarar uma variável no javascript, sendo elas:

- ▶ var.
- ▶ let.
- ▶ const.
- ▶ Não usando nada (Não é recomendado).

A palavra 'var' só é recomendada para navegadores antigos que não conseguiram suportar a atualização do javascript em 2015 quando foram incluídas as palavras 'let' e 'const'.

Variáveis no Javascript: var

A palavra-chave ‘var’ deixou de ser recomendada, pois com ela é possível “criar” a mesma variável várias vezes em diversos lugares do código, o que pode acabar gerando alguns problemas futuros. Por exemplo:

```
var nome = 'Alvaro';
var x = 3;
var nome = 'Felipe';
var nome = 'Isaque';
```

Para isso, ela foi substituída pela palavra-chave ‘let’, cuja mesma não pode “recriar” uma variável já declarada, mas pode atualizar seu valor. Por exemplo:

```
let nome = 'Isaque';
let x = 3;
nome = 'Gutierrez';
```

Variáveis no Javascript: let

O ‘let’, utilizado para substituir a palavra-chave ‘var’, possui algumas diferenças da mesma, sendo elas:

- ▶ Ele não pode recriar uma variável, seu código irá apresentar um erro caso tente.
- ▶ Ele cria variáveis do tipo bloco, em resumo, caso você crie um ‘let’ dentro de uma função, não conseguirá acessá-lo em nenhuma parte do código além daquela função.
- ▶ Ao contrário do var, não é possível usar uma variável antes de declará-la.

Variáveis no Javascript: const

O terceiro tipo de variável, é o mais único deles, pois o ‘const’ não pode ser re-declarado (assim como ‘let’), nem ter seu valor alterado. Ele é usado quando for armazenar um valor inalterado, como PI, ou regras.

Observações sobre o const:

- ▶ Não é possível declarar um ‘const’ e atribuir o valor em outra linha.
- ▶ É muito utilizado em arrays, funções, objetos, etc.
- ▶ Assim como ‘let’, ‘const’ possui um escopo de bloco.
- ▶ Não é possível usar uma variável constante antes de declará-la.

Variáveis no Javascript: variável indefinida

Caso você declare uma variável, e logo em seguida acabe colocando um ponto e vírgula (;) antes mesmo de ter atribuído um valor. Esta mesma variável receberá o valor ‘undefined’ que significa ‘indefinido’.

```
let quantidade_projetos;  
document.getElementById("demo").innerHTML = quantidade_projetos;
```

undefined

OBS: Caso você vá desenvolver o hábito de criar variáveis indefinidas para depois alterar seus valores no decorrer do código, tenha **MUITO** cuidado caso esteja criando códigos ligados a lógica de programação. Lembro que passei por um problema absurdo ao tentar usar uma variável indefinida como uma variável contador, neste caso, armazene o valor 0 nela para que se torne numérica.

Variáveis no Javascript: exibindo elas

Para exibir uma variável ou escrevê-la na página do seu site, basta usar uma das 4 formas vistas anteriormente, como a 'innerHTML'. Por exemplo:

```
<p id="demo"></p>

<script>
let tenente = "Lucas";
document.getElementById("demo").innerHTML = tenente;
</script>
```

Lucas

Explicando este código de forma rápida, o 'document' representa o 'body' do HTML, e traduzindo a grosso modo, 'getElementById' = 'pegar o elemento pelo ID',

Variáveis no Javascript: multi-declaração

É possível declarar diversas variáveis de forma seguida apenas as separando por vírgula, por exemplo:

```
let tenente = "Lucas", code_ranger = "Alvaro", foca = "Felipe";
document.getElementById("demo").innerHTML = foca;
```

Felipe

Também pode ser escrito da seguinte forma:

```
let jailson_saia = "Karen",
braco_curto = "Isaque",
code_ranger = "Freitas";
document.getElementById("demo").innerHTML = braco_curto;
```

Isaque

Concatenando textos com números

Ao usar o operador de soma (+) em números, acarretará na soma dos mesmos.

Mas, quando usado o operador de soma (+) em textos, eles serão concatenados, também pode ser usado para concatenar texto com variável.

No entanto, o que aconteceria caso tente usar o operador de soma para textos e números? como no exemplo a seguir:

```
let x = "5" + 2 + 3;
```

Na verdade, o que acontece é que ao tentar somar números com textos, todos os componentes acabam se transformando em textos também, tendo o seguinte resultado:

523

Os números foram concatenados.

Operadores no Javascript:

Existem alguns tipos de operadores no javascript, sendo usados de jeitos diferentes. Os operadores são:

- ▶ Operadores aritméticos.
- ▶ Operadores de atribuição.
- ▶ Operadores de comparação.
- ▶ Operadores lógicos.
- ▶ Operadores condicionais.
- ▶ Operadores de tipo.

Operadores no Javascript: aritméticos

Operator	Description
+	Adição
-	Subtração
*	Multiplicação
**	Exponenciação (ES2016)
/	Divisão
%	Módulo (Division Remainder)
++	Aumento
--	Decréscimo

Incremento e decremento no Javascript

O incremento, assim como o decremento, são indispensáveis no uso dos loops (será mostrado mais tarde).

O incremento é o simples ato de somar mais 1, nos loops ele serve para ficar adicionando +1 na variável e impedir de deixá-la parada, fazendo o loop ficar infinito.

Ele pode ser escrito de duas formas, sendo elas:

- ▶ `variavel++` = caso o use fora de um loop, ele irá apresentar o valor original da variável, e depois irá acrescentar +1.
- ▶ `++variavel` = Ele é o oposto do exemplo acima, pois ele irá incrementar o +1 da variável para depois exibi-la.

Incremento e decremento na prática.

variavel++.

```
let numero = 2; // Declarando variável com valor 2
document.write("O valor da variável numero é: " + numero++ ); // Exibindo
variável com valor 2 e depois acrescentando +1.

document.write("<br>" + " Agora seu valor é: " + numero); // Exibindo
variável que agora tem valor 3
```

O valor da variável numero é: 2
Agora seu valor é: 3

++variavel.

```
let numero = 2; // Declarando variável com valor 2
document.write("O valor da variável numero é: " + ++numero ); // Incrementado +1 na variável e depois exibindo.

document.write("<br>" + " Agora seu valor é: " + numero); // Exibindo o
valor atual da variável
```

O valor da variável numero é: 3
Agora seu valor é: 3

OBS: O processo é o mesmo com o decremento, a diferença é que ele subtrai 1 ao invés de somar.

Operadores no Javascript: atribuição

Operador	Exemplo	O mesmo que
=	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code>
<code>**=</code>	<code>x **= y</code>	<code>x = x ** y</code>

Para facilitar o entendimento, vamos supor que X possua o valor 5, então ao escrever o código '`x += y`' é o mesmo que dizer: '`x = 5 + y`'. O mesmo se aplica aos outros exemplos.

Operadores no Javascript: comparação

Operador	Descrição
<code>==</code>	igual a
<code>====</code>	igual valor e tipo igual
<code>!=</code>	não é igual
<code>!==</code>	não igual valor ou não igual tipo
<code>></code>	maior que
<code><</code>	menos de
<code>>=</code>	maior ou igual a
<code><=</code>	menor ou igual a
<code>?</code>	operador ternário

OBS: Será explicado a respeito do operador ternário mais pra frente.

Operadores no Javascript: operador igual (==)

O operador igual serve para verificarmos se um valor é igual a outro.

Por exemplo:

`2 == 2` (2 é igual a 2, logo, é verdadeiro).

`2 == 3` (2 não é igual a 3, logo, é falso).

Ele não diferencia o tipo de variável. Logo:

`2 == "2"` (2 é igual a “2”, é verdadeiro, pois ambos possuem o mesmo número apesar de serem de tipos diferentes).

`2 == "b"` (2 não é igual a “b”, logo, é falso).

Operadores no Javascript: operador idêntico (===)

Ao contrário do exemplo visto anteriormente, o operador idêntico difere o tipo de dado. Logo:

`2 === 2` (2 é idêntico a 2, é verdadeiro, pois ambos representam o mesmo número e são do mesmo tipo de dado).

`2 === 3` (2 é idêntico a 3, é falso, pois apesar de possuírem o mesmo tipo de dado, são de números diferentes).

`2 === "2"` (2 é idêntico a "2", é falso, pois apesar de representarem o mesmo número, são de tipos diferentes de dados).

Operadores no Javascript: operador diferente (!=)

Ele faz uma verificação se o valor é diferente de outro valor. Por exemplo:

`2 != 2` (2 é diferente de 2, é falso, pois 2 é igual a 2).

`2 != 3` (2 é diferente de 3, é verdadeiro, pois são números diferentes).

`2 != "2"` (2 é diferente de "2", é falso, pois ele não leva em conta o tipo de dado).

`2 != "a"` (2 é diferente de "a", é verdadeiro).

Operadores no Javascript: operador Não é idêntico (!=)

O operador ‘Não é idêntico’ faz 2 verificações ao mesmo tempo, se um valor é diferente de outro, **OU** se o tipo de valor é diferente de outro. Por exemplo:

2 != “2” (2 não é idêntico a “2”, é verdadeiro, pois apesar de serem o mesmo número, são de tipos **diferentes** de dados).

2 != 2 (2 não é idêntico a 2, é falso, pois 2 é igual a 2).

2 != 3 (2 não é idêntico a 3, é verdadeiro, pois apesar de terem o mesmo tipo de variável, são números **diferentes**).

Operadores no Javascript: operador maior que (>)

O operador ‘maior que’ verifica se o primeiro valor é maior que o segundo. Por exemplo:

`5 > 4` (5 é maior que 4, é verdadeiro).

`5 > 5` (5 é maior que 5, é falso, pois 5 é igual a 5).

`5 > 6` (5 é maior que 6, é falso, pois 5 é menor que 6).

`5 > "3"` (5 é maior que “3”, é verdadeiro, pois apesar de serem tipo de dados diferentes, 5 é maior que 3).

Operadores no Javascript: operador menor que (<)

O operador ‘menor que’ faz o oposto do ‘maior que’ e verifica se o primeiro número é menor que o segundo. Por exemplo:

`5 < 4` (5 é menor que 4, é falso, pois 5 é maior que 4).

`5 < 5` (5 é menor que 5, é falso, pois 5 é igual a 5).

`5 < 6` (5 é menor que 6, é verdadeiro).

`5 < “6”` (5 é menor que 6, é verdadeiro, pois apesar de serem valores de tipos diferentes, 5 é menor que 6).

Operadores no Javascript: operador maior ou igual (\geq)

O operador maior ou igual verifica se o primeiro valor é maior ou igual ao segundo valor. Por exemplo:

$5 \geq 4$ (5 é maior ou igual a 4, é verdadeiro, pois 5 é maior que 4).

$5 \geq 5$ (5 é maior ou igual a 5, é verdadeiro, pois 5 é igual a 5).

$5 \geq 6$ (5 é maior ou igual a 6, é falso, pois 5 não é maior que 6, nem igual).

$5 \geq "5"$ (5 é maior ou igual a “5”, é verdadeiro, pois 5 é igual a “5”, mesmo sendo de tipos diferentes de dados).

$5 \geq "4"$ (5 é maior ou igual a “4”, é verdadeiro, pois 5 é maior que “4”, apesar de serem tipos de dados diferentes).

Operadores no Javascript: operador menor ou igual (\leq)

O operador ‘menor ou igual’ verifica se o primeiro valor é menor ou igual ao segundo. Por exemplo:

$5 \leq 4$ (5 é menor ou igual a 4, é falso, pois 5 é maior que 4).

$5 \leq 5$ (5 é menor ou igual a 5, é verdadeiro, pois 5 é igual a 5).

$5 \leq 6$ (5 é menor ou igual a 6, é verdadeiro, pois 5 é menor que 6).

$5 \leq "5"$ (5 é menor ou igual a “5”, é verdadeiro, pois 5 é igual a “5” mesmo sendo de tipos de dados diferentes).

$5 \leq "6"$ (5 é menor ou igual a “6”, é verdadeiro, pois 5 é menor que 6, mesmo sendo de tipos de dados diferentes).

Operadores no Javascript: operadores lógicos

Operador	Descrição
&&	lógico e
	lógico ou
!	lógico não

Os 2 primeiros operadores verificam 2 ou mais condições ao mesmo tempo, mas possuem comportamentos diferentes em relação a elas.

O operador **&&** (**E**) verifica se todas as condições são verdadeiras, caso uma delas seja falsa, ele retorna falso por completo.

O operador **||** (**OU**) verifica se uma ou mais das condições são verdadeiras, ele só retorna falso caso todas sejam falsas.

O operador **!** (**não**) Nega o estado de uma condição, se for verdadeira ele retorna falsa e vice-versa.

Tipos de dados no Javascript

O javascript possui 8 tipos de dados, sendo eles:

- ▶ string.
- ▶ number.
- ▶ bigint.
- ▶ boolean.
- ▶ undefined.
- ▶ null.
- ▶ symbol.
- ▶ object.

Tipos de dados no Javascript: string

O tipo de dado string é usado em cadeias de caracteres, como textos, números, etc. Sempre estando coberto de aspas simples ou duplas. Por exemplo:

```
let email = 'fulano123@gmail.com';
```

```
let RA = "0001105303839" (obviamente é um número aleatório);
```

Ele é recomendável ser usado em números quando houver a possibilidade do primeiro caractere ser um 0, como o R.A por exemplo. O tipo de dado 'number' ignora a existência do 0 caso ele seja o primeiro caractere, desconsiderando completamente o cadastro do indivíduo.

Tipos de dados no Javascript: number

O tipo de dado number é usado estritamente em números e se divide em dois tipos. Sendo eles:

- ▶ int = números inteiros, sem casas decimais. Ex: 114.
- ▶ float = números com casas decimais, são chamados de pontos flutuantes ou reais (no visualg). Ex: 114.00.

Como foi avisado anteriormente, ele ignora a existência do número 0 caso ele seja o primeiro caractere, então muito cuidado.

Soma e concatenação no Javascript:

Caso você queira somar determinados números antes de concatená-los em uma cadeia de caracteres, pode acabar transformando o tipo de dado dos números em 'string'.

```
document.write('Quanto é 5 + 2: ' + 5 + 2); Quanto é 5 + 2: 52
```

No entanto, você pode simplesmente colocar os números que deseja somar dentro de parênteses.

```
document.write('Quanto é 5 + 2: ' + (5 + 2)); Quanto é 5 + 2: 7
```

Tipos de dados no Javascript: BigInt

O tipo de dado BigInt foi inventado recentemente (2020) e serve para armazenar números que seriam grandes demais para guardá-los em um tipo de dado number. Por exemplo:

```
let x = BigInt("123456789012345678901234567890");
```

Tipos de dados no Javascript: boolean

O tipo de dado 'boolean' pode armazenar apenas 2 valores, sendo 'true(verdadeiro)' ou 'false(falso)'. Além disso, ele serve justamente para realizar comparações, seu uso é indispensável quando for aprender 'desvio condicional'.

```
let x = 5;
let y = 5;
let z = 6;
(x == y)      // Returns true
(x == z)      // Returns false
```

Tipos de dados no Javascript: array

Arrays são escritos entre colchetes ([]) e podem ser lidos como listas no Javascript. Por exemplo:

```
const cars = ["Saab", "Volvo", "BMW"];
```

O exemplo acima declara uma variável carro que possui 3 valores.

Como pode ver, ele é **extremamente** recomendado para se aprender, pois pode estar economizando uma quantidade surreal de espaço, por estar armazenando diversos valores em uma variável ao invés de criar “diversas gavetas para guardar uma coisa cada”.

OBS: Será aprendido mais sobre arrays no decorrer da apostila.

Tipos de dados no Javascript: object

Nos objetos, os dados armazenados são divididos entre 'propriedades' e 'valores'. Além disso, eles são escritos dentro de chaves ({}). Por exemplo:

```
const pessoa = {  
    nome: "Jailson",  
    apelido: "Braço Curto",  
    titulo: "Melhor professor do PROA",  
    responsavel: "Por traumatizar milhares de alunos com step by step"  
};  
  
document.getElementById("demo").innerHTML =  
'O nome do professor é: ' + pessoa.nome + "<br>" +  
'Seu apelido é: ' + pessoa.apelido + "<br>" +  
'Seu titulo é: ' + pessoa.titulo + "<br>" +  
'Ele é responsável por: ' + pessoa.responsavel;
```

O nome do professor é: Jailson
Seu apelido é: Braço Curto
Seu titulo é: Melhor professor do PROA
Ele é responsável por: Por traumatizar milhares de alunos com step by step

Tipos de dados no Javascript: indefinidos

Variáveis são dadas como indefinidas após serem declaradas e não receberem valor algum. Por exemplo:

```
let vazio;
```

É um estágio onde ele não possui nenhum tipo de valor. Um aviso é para tomar cuidado caso você crie o hábito de criar variáveis indefinidas para só depois atribuir valores a elas, pois você pode acabar recebendo um erro de operação como por exemplo:

Caso use uma variável indefinida como ‘contadora’ começando com 0, seu código apresentará um erro, pois a variável indefinida não possui valor numérico, logo, é impossível usá-la como contadora.

Tipos de dados no Javascript: falso indefinido

Como expliquei anteriormente, uma variável indefinida é dada assim pois não possui nenhum valor, então ela não continua nesse estado após receber um valor. Para esclarecimento, irei dar 2 exemplos de falsos indefinidos, que são eles:

```
let vazio = ""; // A variável não é indefinida, pois os "" a tornam uma
string
document.write(typeof vazio);
```

```
let vazio = 0; // A variável não é indefinida, pois os 0 conta como um
número
document.write(typeof vazio);
```

Funções no Javascript

Uma função é um bloco de código desenvolvido para realizar determinada ação, por exemplo: **multiplicar**, somar, etc.

```
function multiplicar(numero1, numero2) {  
    return numero1 * numero2;  
}  
document.write(multiplicar(5, 3));  
// 15
```

Para esclarecimento de dúvidas, a função só pode ser exibida quando seu nome (palavra atrás dos parênteses) for escrito, juntamente com os parênteses, ela é extremamente útil para reaproveitamento de código, pois você o constrói uma vez e sempre que precisar basta chamá-la de novo.

Funções no Javascript: parâmetros

Os parâmetros são as palavras que ficam entre parênteses quando a função estiver sendo declarada.

```
multiplicar(numero1, numero2)
```

Essas palavras atuam como variáveis, e seus valores podem ser atribuídos fora da função, como:

```
document.write(multiplicar(5, 3));
```

Após os valores, serem dados como parâmetros, a função os substitui por estes números em todo seu bloco de código, no caso:

```
return numero1 * numero2;
```

OBS: No exemplo acima, a função irá retornar 15.

Funções no Javascript: chamando função

As funções podem ser chamadas de 3 jeitos, sendo eles:

- ▶ Quando usuário realiza determinada ação (aperta um botão por exemplo).
- ▶ Quando é chamada pelo próprio código javascript (como no exemplo anterior).
- ▶ Quando é auto-chamada.

No caso de apertar o botão, é possível através do atributo ‘onclick’ no próprio código HTML, onde se coloca o nome da função junto com parênteses.

```
<button onclick="multiplicar()">Clique aqui!</button>
```

Funções no Javascript: return

Caso você vá fazer alguma coisa que exija um retorno e escolha usar uma função (pois é recomendado), o return será indispensável, pois normalmente ele é o resultado da operação (**OBS:** estou usando operação como exemplo, mas é qualquer código que vá retornar algo).

Por exemplo: se você escrever o código ‘z = x + y;’, o Z será o valor que irá retornar.

Não é obrigatório que as funções possuam retorno, mas para ajudar, vamos fingir que eles possuem tipos, para dizer que a função é do tipo string, ela deve **retornar** um valor do tipo string. Ex: ‘return nome’.

A função anterior como ‘multiplicar’ seria do tipo number.

Funções no Javascript: por que usar funções?

As funções são extremamente úteis, principalmente para reaproveitamento de código, uma vez que você só precisa construí-las uma vez para reutilizar em todo o código, apenas mudando seus valores como parâmetros. Por exemplo:

```
function calcular_fahrenheit(celsius) {  
    let fahrenheit = celsius/5 * 9 + 32;  
    return fahrenheit  
}  
document.write( calcular_fahrenheit(100) )
```

Esta função, por exemplo, poderá ser usada quantas vezes quiser apenas escrevendo seu nome e passando a temperatura em graus celsius como parâmetro.

Funções no Javascript: sem parênteses

Ao escrever uma função sem parênteses em um dos 4 tipos de saídas, ela irá escrever em uma única linha todo o código contido na função. Por exemplo na função vista anteriormente (`calcular_fahrenheit`):

```
document.write( calcular_fahrenheit ); function calcular_fahrenheit(celsius) { let fahrenheit = celsius/5 * 9 + 32; return fahrenheit }
```

Pode ver que na saída, as linhas de código são separadas por ponto e vírgula.

Funções no Javascript: variáveis

Caso você perceba que está usando muito um certo valor como parâmetro da função, pode armazená-lo em uma variável para facilitar a consulta. Dessa forma:

```
let f = calcular_fahrenheit(100);
document.write('A temperatura 100°C em fahrenheit é: ' + f + "°F");

```

Claro que não é obrigatório, podendo passar o valor da função de forma direta:

```
document.write('A temperatura 100°C em fahrenheit é: ' +
calcular_fahrenheit(100) + "°F");
```

Funções no Javascript: variáveis locais

Agora podemos ver um exemplo claro do 'let'.

Uma variável global, só pode ser acessada no escopo em que foi criada, um escopo é tudo aquilo que estiver entre chaves ({}).

Dessa forma, uma variável que foi criada dentro de uma função, não pode ser acessada fora desta função.

```
function titulos() {  
    let braco_curto = 'Felipe';  
    // Pode usar a variável 'braco_curto'  
}  
// Não pode usar a variável 'braco_curto'|
```

Objetos no Javascript

Na vida real, um carro é um objeto. Pois ele possui propriedades como peso e cor, além de possuir métodos como iniciar e parar.

Objeto	Propriedades	Métodos
	<code>car.name = Fiat</code> <code>car.model = 500</code> <code>car.weight = 850kg</code> <code>car.color = branco</code>	<code>car.start()</code> <code>car.drive()</code> <code>car.brake()</code> <code>car.stop()</code>

Objetos no Javascript

Objetos também são variáveis, mas podem conter muitos valores, além de conseguir especificar eles com as propriedades.

Em objetos, quebras de linha e espaço não fazem diferença no resultado. Por exemplo este caso:

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
  
const person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50,  
    eyeColor: "blue"  
};
```

Objetos no Javascript: acessando propriedades

No Javascript, é possível acessar as propriedades dos objetos de duas maneiras:

Escrever o nomeObjeto.nomePropriedade.

```
person.lastName;
```

Escrever o nomeObjeto["nomePropriedade"].

```
person["lastName"];
```

Objetos no Javascript: métodos

Um objeto também pode ter métodos, que no caso são ações que você manda o computador fazer, seja calcular valores, concatená-los, etc. Ele é escrito em forma de uma função sem nome. Por exemplo:

```
const pessoa = {
    primeiroNome: 'Alvaro',
    idade: undefined,
    segundoNome: 'Carvalho de Lima',
    nomeCompleto: function() {
        return this.primeiroNome + ' ' + this.segundoNome;
    }
}
```

Para chamar o método, basta fazer igual a propriedades normais, porém adicionando os parênteses na frente. Por exemplo:

```
document.write( pessoa.nomeCompleto() )
```

OBS: Por enquanto, pense no 'this' como o método usado para transformar e acessar as propriedades fora de seu escopo.

Eventos no Javascript

Explicando de forma simplificada, eventos são “coisas” que acontecem na página HTML e fazem o javascript “reagir” a esses eventos.

Alguns exemplos de eventos são:

- ▶ Quando a página acabou de carregar.
- ▶ Quando um botão foi clicado.

O javascript permite que você execute código quando estes eventos são detectados. Para isso, basta escrever:

```
<elemento nomeEvento=" código Javascript ">
```

No caso do botão clicado:

```
<button onclick="window.alert("Eae povão que tem projeto pra fazer")">Lembrete</button>
```

www.w3schools.com diz

Eae povão que tem projeto pra fazer

OK

Eventos no Javascript: exemplos

O Javascript pode perceber alguns eventos do HTML, sendo eles:

- ▶ Fazer determinada ação quando a página é carregada.
- ▶ Fazer determinada ação quando a página é fechada.
- ▶ Ação que deve ser executada quando o usuário aperta um botão.
- ▶ Conteúdo que deve ser verificado quando um usuário insere dados.

Nos atributos HTML é possível usar seguintes tipos de código Javascript:

- ▶ Atributos de evento HTML podem executar código javascript.
- ▶ Atributos de evento HTML podem chamar funções javascript.

Eventos no Javascript: eventos comuns

Acontecimento	Descrição
onchange	Um elemento HTML foi alterado
onclick	O usuário clica em um elemento HTML
onmouseover	O usuário move o mouse sobre um elemento HTML
onmouseout	O usuário afasta o mouse de um elemento HTML
onkeydown	O usuário pressiona uma tecla do teclado
onload	O navegador terminou de carregar a página

String no Javascript:

Um string nada mais é do que uma cadeia de caracteres que esteja dentro de aspas simples ou duplas.

É possível usar os 2 tipos de aspas em uma mesma frase, basta tratá-los como parênteses, por exemplo:

```
let resposta1 = "Seu nome é 'Jailson"'; Seu nome é 'Jailson'  
let resposta2 = 'Seu nome é "Obama"'; Seu nome é "Obama"
```

Porém, caso você tente usar o mesmo tipo de aspas, pode acabar gerando um erro. Como:

```
let texto = "O nome dele é "Jailson", O nome dele é "Obama."
```

String no Javascript: uso da barra invertida

Para impedir que uma aspa dupla acabe fechando a que começou o texto, pode usar uma barra invertida (\) antes da aspa. Por exemplo:

Código	Resultado	Descrição
\'	'	Cotação única
\\"	"	Aspas duplas
\\\	\	Barra invertida

Por exemplo:

```
let texto = "O nome dele é \"Jailson \\", O nome dele é \"Obama\"."
          O nome dele é "Jailson ", O nome dele é "Obama".
```

String no Javascript: alguns códigos que usam barra invertida

Código	Resultado
\b	Backspace
\f	Feed de formulários
\n	Nova Linha
\r	Retorno de Carro
\t	Guia horizontal
\v	Guia Vertical

O \n é muito usado para quebra de linha, assim como o
.

String no Javascript: quebra de linha nos textos

Foi falado antes que caso criasse o hábito de quebrar as linhas de código para deixá-lo mais legível, seria recomendado fazê-lo no operador de atribuição (=). Por exemplo:

```
document.getElementById("demo").innerHTML =  
"Hello Dolly!";
```

Mas caso você tenha desenvolvido ‘toc’ e queira insistir na quebra mesmo estando no meio do texto, basta usar uma barra invertida (\) antes de quebrar a linha, dessa forma:

```
document.getElementById("demo").innerHTML = "Hello \  
Dolly!";
```

String no Javascript: quebra de linha nos textos

Porém, o método com a barra invertida não é das mais recomendadas devido sua falta de suporte e alguns navegadores não permitem seu uso, então você pode simplesmente concatenar o texto com a palavra que sofrerá a quebra de linha. Dessa forma:

```
document.getElementById("demo").innerHTML = "Hello " +  
    "Dolly!";
```

A barra invertida não pode ser usada antes do texto, pois acarretará em diversos erros.

Métodos de string no Javascript

Será apresentado alguns métodos para utilizar em strings. Estes métodos podem aplicar diversas funções nelas, desde saber quantos caracteres possui o texto em que digitou, até a trocar uma palavra por outra neste mesmo texto, verá mais desses métodos a seguir:

A sintaxe da maioria dos métodos basicamente é:

nomeVariavel.nomeMétodo(parametros)

Métodos de string no Javascript: length

Para checar quantos caracteres foram escritos em uma variável, basta usar a propriedade 'length'. Por exemplo:

```
let text = "ABCDEFGHIJKLMNPQRSTUVWXYZ";  
let length = text.length;
```

26

Métodos de string no Javascript: slice()

O método slice() consegue estar recortando uma palavra, frase, ou até mesmo parte de um texto, após informar dois parâmetros, sendo eles:

slice(**parametro1**, **parametro2**) = O parâmetro 1 refere-se a **posição** do caractere em que irá começar a recortar, já o segundo parâmetro, se refere a **posição** final do caractere. Por exemplo:

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(7, 13);
```

A palavra armazenada na variável ‘part’ será ‘Banana’, pois a letra B se encontra no 7º caractere, e a letra ‘a’ é o 13º.

OBS: No Javascript, a **primeira** posição é sempre 0.

Métodos de string no Javascript: slice()

Caso seja informado apenas o primeiro parâmetro, ele começará a recortar as palavras a partir do caractere informado. Por exemplo:

```
text.slice(7); Banana, Kiwi
```

Caso seja informado um valor negativo, ele irá inverter a ordem (no caso o último caractere será o 0), além de percorrer até o número informado, por exemplo:

```
let text = "Apple, Banana, Kiwi"; e, Banana, Kiwi
let part = text.slice(-15);
```

Métodos de string no Javascript: slice()

Caso seja informado 2 valores negativos, o primeiro valor irá dizer a onde começará a recortar, e o segundo indicará onde vai acabar.
Por exemplo:

```
let text = "Apple, Banana, Kiwi"; pple  
let part = text.slice(-18, -14);
```

Métodos de string no Javascript: substring()

O método ‘substring’ é similar a ‘slide’, sua principal diferença é que ela não trata de valores abaixo de 0.

```
let str = "Do pescoco pra baixo é canela!";
document.getElementById("demo").innerHTML = str.substring(21, 30);
```

é canela!

Caso informe apenas 1 valor, ele começará a cortar a partir da posição informada até o texto acabar.

```
let str = "É no desconforto que se cresce";
document.getElementById("demo").innerHTML = str.substring(8);
```

conforto que se cresce

Métodos de string no Javascript: substr()

O método ‘substr’ é semelhante a ‘slice’, a diferença é que o segundo parâmetro remete à quantidade de caracteres que serão cortados, e não a posição que o corte será terminado. Por exemplo:

```
let str = "Ta fácil? Já terminou os projetos?";
document.getElementById("demo").innerHTML = str.substr(25, 8); projetos
```

Se o parâmetro for negativo, a posição começará a partir do final da string. Por exemplo:

```
let str = "Ta fácil? Já terminou os projetos?";
document.getElementById("demo").innerHTML = str.substr(-21); terminou os projetos?
```

Métodos de string no Javascript: replace()

O método replace serve para trocar uma palavra na cadeia de caracteres, é extremamente útil em jogos para menores de idade, simplesmente porque é com ela que é feita a censura de certas palavras por ****.

Sua sintaxe é: nomeVariavel.replace("palavraDoTexto", "novaPalavra").

Por exemplo:

```
let str = "Ta fácil? Já terminou os projetos?";
document.getElementById("demo").innerHTML = str.replace("fácil", "doce")
```

Ta doce? Já terminou os projetos?

OBS: Este método substitui apenas a primeira palavra informada, no caso se houvesse 2 “fácil”, ele iria substituir apenas o primeiro por “doce”.

Métodos de string no Javascript: replace()

Por padrão, o replace é ‘case sensitive’, podendo diferenciar maiúsculas de minúsculas. Dessa forma, escrever “FÁCIL” apresentaria um erro no exemplo anterior. Por exemplo:

```
let str = "Ta fácil? Já terminou os projetos?";
document.getElementById("demo").innerHTML = str.replace("FÁCIL", "doce")
```

Ta **fácil?** Já terminou os projetos?

Contudo, por incrível que pareça há uma forma de contornar isso, usando uma expressão regular com o sinalizador /i. Por exemplo:

```
let str = "Ta fácil? Já terminou os projetos?";
document.getElementById("demo").innerHTML = str.replace(/FÁCIL/i, "doce")
```

Ta **doce?** Já terminou os projetos?

OBS: A expressão regular /i significa algo em torno de “insensitive” (insensível).

Métodos de string no Javascript: replace()

Caso houvesse mais de uma palavra que o 'replace' tivesse que substituir, ele trocaria apenas a primeira delas, por exemplo:

```
let str = "Se o bug buga tudo buga tudo ou somente buga ou bugou mesmo?";
document.getElementById("demo").innerHTML = str.replace("buga", "confunde")
Se o bug confunde tudo buga tudo ou somente buga ou bugou mesmo?
```

Porém, uma forma de resolver isso é com o uso da expressão regular com o sinalizador /g. Por exemplo:

```
let str = "Se o bug buga tudo buga tudo ou somente buga ou bugou mesmo?";
document.getElementById("demo").innerHTML = str.replace(/buga/g, "confunde")
Se o bug confunde tudo confunde tudo ou somente confunde ou bugou mesmo?
```

OBS: A expressão regular /g significa algo em torno de “global”.

Métodos de string no Javascript: replaceAll()

Caso você não queira usar a expressão regular global (/g). O javascript adicionou recentemente (2021) um novo método chamado 'replaceAll' que permite fazer a mesma coisa, que é substituir todas as palavras de uma só vez.

```
let str = "Se o bug buga tudo buga tudo ou somente buga ou bugou mesmo?";
document.getElementById("demo").innerHTML = str.replaceAll("buga", "confunde")
Se o bug confunde tudo confunde tudo ou somente confunde ou bugou mesmo?
```

No entanto, por ser adicionado recentemente, ele não funcionará em navegadores mais antigos como o internet explorer.

Métodos de string no Javascript: toUpperCase() e toLowerCase()

O método 'toUpperCase' serve para transformar todos os caracteres da string em letras maiúsculas. Por exemplo:

```
let str = "Vai mesmo atravessar a rua pra pisar em casca de banana?";  
document.getElementById("demo").innerHTML = str.toUpperCase();  
VAI MESMO ATRAVESSAR A RUA PRA PISAR EM CASCA DE BANANA?
```

O método 'toLowerCase' faz o oposto e serve para deixar todos os caracteres em letras minúsculas

```
let str = "Acabou A Criatividade Para Fazer Esses Textos";  
document.getElementById("demo").innerHTML = str.toLowerCase();  
acabou a criatividade para fazer esses textos
```

Métodos de string no Javascript: concat()

O método 'concat' serve para concatenar 1 ou mais palavras, sua sintaxe é:

```
nomeVariavel.concat("palavraConcatenada",
"palavraConcatenada2", "etc").
```

Por exemplo:

```
let medo = "Da um exemplo";
let frase = medo.concat(' ', "de um framework pra mim")
document.write(frase)
```

Da um exemplo de um framework pra mim

OBS: Sim, usar o operador de soma (+) não tem diferença alguma, **contudo**, todavia, entretanto.. O método concat pode ser usado para automatizar um processo, por exemplo, sempre que a pessoa apertar o botão 'entediado (não olha pra mim, acabou a criatividade)' as frases dele irão terminar com 'que tédio...'.

Métodos de string no Javascript: padStart() e padEnd()

Os métodos ‘padStart’ e ‘padEnd’ servem para facilitar o adicionamento de caracteres, com as palavras ‘start’ e ‘end’ indicando a posição em que os caracteres serão adicionados. Sua sintaxe é:

nomeVariavel.padStart((posição do caractere em que será adicionado), “caractere ou palavra que irá preencher”).

Essa sintaxe também se aplica ao ‘padEnd’. Veja um exemplo para facilitar a compreensão:

```
let pedido = "Eu quero um hamburguer tamanho ";
document.getElementById("demo").innerHTML = pedido.padEnd(35,"x");
```

Eu quero um hamburguer tamanho xxxx

```
let pedido = "Eu quero um hamburguer tamanho ";
document.getElementById("demo").innerHTML = pedido.padStart(35,"x");
```

xxxxEu quero um hamburguer tamanho

Métodos de String no Javascript: charAt()

O método charAt() retorna o caractere em que tiver sua posição especificada. Veja um exemplo para facilitar a compreensão:

```
var text = "Quero café";
document.getElementById("demo").innerHTML = text.charAt(0); Q
```

```
var text = "Quero café";
document.getElementById("demo").innerHTML = text.charAt(7); a
```

Outra forma de fazer isso, porém que não é muito recomendada, é substituir o 'charAt' por colchetes ([]). Por exemplo:

```
var text = "Quero café";
document.getElementById("demo").innerHTML = text[7]; a
```

Métodos de pesquisa em cadeias de caracteres: indexOf()

O método indexOf() procura pela palavra especificada entre os parênteses e retorna a posição do primeiro caractere da palavra. Por exemplo:

```
var text = "Por favor, encontre a palavra 'encontre': ";
document.getElementById("demo").innerHTML = text.indexOf('encontre'); 11
```

Caso tenha contado a posição de cada caractere, deve ter reparado que na verdade o 'indexOf' inicia a contagem é 0. Além disso, caso seja informado uma palavra que é encontrada várias vezes no texto, o 'indexOf' irá procurar a posição apenas da primeira palavra que ele encontrar.

Métodos de pesquisa em cadeias de caracteres: lastIndexOf()

Se o método 'indexOf' busca sempre a primeira palavra do texto, caso haja mais de uma. O 'lastIndexOf' busca a **última** palavra do texto que é informada, no caso anterior o 'encontre' **começava** na posição 11, agora veja em qual ele começa:

```
var text = "Por favor, encontre a palavra 'encontre': ";
document.getElementById("demo").innerHTML = text.lastIndexOf('encontre'); 31
```

Tanto para o 'indexOf' quanto para o 'lastIndexOf', irão retornar '-1' caso a palavra informada entre parênteses não seja encontrada no texto:

```
var text = "Já fez os projetos?";
document.getElementById("demo").innerHTML = text.indexOf('fiz'); -1
```

Métodos de pesquisa em cadeias de caracteres: lastIndexOf()

Tanto ele quanto o ‘indexOf’ aceitam um segundo parâmetro, no caso sendo para informada de onde podem começar a procurar. Por exemplo:

```
let text = "Por favor, encontre a palavra 'encontre'";
document.getElementById("demo").innerHTML = text.indexOf("encontre", 20);31
```

O mesmo irá acontecer com o ‘lastIndexOf’, a diferença é que ele começa a procurar de trás para frente.

Métodos de pesquisa em cadeias de caracteres: search()

Similar ao método ‘indexOf’, o search pode procurar a posição em que a palavra informada começa. Por exemplo:

```
let text = "Por favor, encontre a palavra 'encontre'";
document.getElementById("demo").innerHTML = text.search("encontre"); 11
```

Porém, há algumas diferenças entre o ‘indexOf’ e o ‘search’. Sendo elas:

- ▶ ‘search’ não é forte o bastante para receber o segundo parâmetro.
- ▶ ‘indexOf’ não consegue procurar uma palavra dentro de expressão regular como o search (dentro de / ao invés de aspas).

Métodos de pesquisa em cadeias de caracteres: includes()

Este método retorna ‘true’ (verdadeiro) se a palavra informada estiver no texto. Logo, ele retorna ‘false’ caso a palavra não seja encontrada. Por exemplo:

```
let text = "Não deixe o mar te engulir";
document.getElementById("demo").innerHTML = text.includes("mar"); true
```

Ele também pode receber um segundo parâmetro, indicando em que posição ele irá começar a busca. Por exemplo:

```
let text = "Não deixe o mar te engulir";
document.getElementById("demo").innerHTML = text.includes("mar", 20); false
```

Nesse caso, ele retornou o valor ‘falso’ porque a palavra mar não está depois da 20º posição.

OBS: O ‘includes’ não é suportado pelo internet explorer.

Método de pesquisa de cadeias de caractere: startsWith()

O método 'startsWith' serve para verificar se a primeira palavra do texto começa com a palavra informada. Por exemplo:

```
let text = "Atravessar a rua para pisar em casca de banana.";
document.getElementById("demo").innerHTML = text.startsWith("Atravessar"); true
```

Ele também aceita um segundo valor para especificar onde ele vai começar a verificar.

O método 'endsWith' serve da mesma forma, tendo como diferença que ele verifica qual é a última palavra do texto.

OBS: Ambos não são suportados no internet explorer.

Literais de modelo no Javascript

Literal de modelo é uma versão alternativa da cadeia de caracteres, pois ele também é do tipo string, sua principal diferença é que seus caracteres estão encobertos com crase (` `). Ele possui alguns sinônimos, como:

- ▶ Literais de modelo
- ▶ Cadeias de caracteres de modelo
- ▶ Modelos de cadeia de caracteres

Veja um exemplo de sua sintaxe:

```
let text = `Hello World!`;
```

OBS: Dentro dele é possível usar os dois tipos de aspas também.

Literais de modelo no Javascript

Ele possui algumas vantagens, por exemplo o fato de que ele é uma cadeia de caracteres multi-linhas, ao contrário das aspas normais que precisam usar a barra invertida para poderem quebrar a linha.

Exemplo de um literal multi-linha.

```
let text =  
  `Não deixe o mar te engolir  
  A segunda onda bate no joelho  
  A terceira onda bate no cóccix  
  A quarta onda ta batendo no peitoral  
  A quinta onda já ta no pescoço`;  
  
document.getElementById("demo").innerHTML = text;
```

Não deixe o mar te engolir A segunda onda bate no joelho A terceira onda bate no cóccix A quarta onda ta batendo no peitoral A quinta onda já ta no pescoço

Literais de modelo no Javascript: variáveis

Outra de suas vantagens é que ele oferece uma forma melhor de intercalar texto com variáveis, que é colocando dentro da cadeia de caracteres(entre as crases). Veja um exemplo para entender mais fácil:

```
let nome = 'Alvaro';
let sobrenome = 'Carvalho de Lima';

let apresentacao = `Meu nome é ${nome} ${sobrenome}, prazer em conhecê-los!`
```

Meu nome é Alvaro Carvalho de Lima, prazer em conhecê-los!

Colocando as variáveis dentro de chaves ({}) atrás de um cífrão (\$), permite que elas sejam escritas junto ao texto, sem necessidade de concatenação.

Literais de modelo no Javascript: operações

Os literais permitem até mesmo usarem operações matemáticas dentro das crases. Veja um exemplo para facilitar a compreensão:

```
let nota1 = 7;  
let nota2 = 9;  
let media = `A média do aluno foi de: ${((nota1 + nota2) / 2)}`;  
  
document.getElementById("demo").innerHTML = media;
```

A média do aluno foi de: 8

Além do \${}, a conta deve ficar dentro de parênteses para prevenção de erro.

OBS: Infelizmente, ele não é suportado no Internet Explorer.

Números no Javascript

Ao contrário dos outros que ocupam espaços diferentes para cada tipo de número, o javascript ocupa 64 bits para cada número.

Agora para os espaçamentos:

Números inteiros (sem casas decimais) podem armazenar um total de 15 números sem erro algum.

Já com números decimais, conseguem estar armazenando um total de 17 números.

```
let x = 0.2 + 0.1;  
document.getElementById("demo").innerHTML = "0.2 + 0.1 = " + x; 0.2 + 0.1 = 0.30000000000000004
```

Números no Javascript: operação com float

Além disso, no javascript a matemática com números decimais (float) não é 100% exata, veja um exemplo:

```
let x = 0.2 + 0.1;  
document.getElementById("demo").innerHTML = "0.2 + 0.1 = " + x; 0.2 + 0.1 = 0.30000000000000004
```

Uma solução é transformar os decimais em inteiros ao multiplicá-los por 10, depois transformar o resultado da operação em decimal novamente, o dividindo por 10. Por exemplo:

```
let y = (0.2*10 + 0.1*10) / 10;  
document.getElementById("demo2").innerHTML = "0.2 + 0.1 = " + y; 0.2 + 0.1 = 0.3
```

Números no Javascript: operações

Com exceção do operador de adição (+), o javascript consegue realizar qualquer operação que dependa de outro operador, por exemplo:

```
let x = "100";
let y = "10";
let z = x - y;
document.getElementById("demo").innerHTML = z;
```

90

Números no Javascript: Not a Number (NaN)

O tipo de variável NaN ocorre quando tentamos fazer uma operação entre números e uma cadeia de caracteres, por exemplo:

```
document.getElementById("demo").innerHTML = 40 / 'projetos' NaN
```

Porém, caso o conteúdo da cadeia de caracteres seja apenas números, a operação irá funcionar, como visto no exemplo anterior e neste aqui:

```
document.getElementById("demo").innerHTML = 40 / 10; 4
```

OBS: Ele também pode ser declarado como valor de variável: `let x = NaN;`

Números no Javascript: not a number (NaN)

Por incrível que pareça, caso vá conferir que tipo de dado é o 'Not a Number', vai aparecer que ele é number..... Enfim, a hipocrisia.

```
let x = NaN;  
document.getElementById("demo").innerHTML = typeof x; number
```

Números no Javascript: infinity

Caso o valor de variável apareça como ‘infinity’, significa que o número da operação é maior do que o limite de números (15 números inteiros e 17 decimais), ou que ele realmente é infinito e não para de aumentar.

```
document.getElementById("demo").innerHTML = 2.6815615859885194e+154 * Infinity  
2.6815615859885194e+154;
```

Outra forma de gerar o valor ‘infinity’ é dividindo qualquer número por 0:

```
let x = 2 / 0;  
document.getElementById("demo").innerHTML = x; Infinity
```

OBS: Assim como NaN, ele é do tipo Number.

Números no Javascript: isInteger()

Este método foi adicionado recentemente e permite verificar se o número informado entre parênteses é inteiro ou não. Por exemplo:

```
document.getElementById("demo").innerHTML =  
Number.isInteger(10) + "<br>" +  
Number.isInteger(10.5);
```

true false

Números no Javascript: isSafeInteger()

Também adicionada recentemente, permite fazer uma verificação sobre o número inteiro ser um “inteiro seguro” que é basicamente qualquer inteiro abaixo do limite em que o javascript consegue trabalhar com precisão, que é: ‘9007199254740991’.

Então, por exemplo, o número 9007199254740991 é seguro, mas o número 9007199254740992 não é.

BigInt no Javascript

Caso precise fazer operações e utilizar números cujo são maiores que o limite do javascript, basta transformá-los em BigInt, colocando um 'n' no final do número, ou usando a função BigInt(número). Por exemplo:

```
let x = 1234567890123456789012345n;  
let y = BigInt(1234567890123456789012345)
```

É possível fazer operações com o BigInt, desta forma:

```
let x = 9007199254740995n;  
let y = 9007199254740995n;  
let z = x * y;  
  
document.getElementById("demo").innerHTML = z;
```

81129638414606735738984533590025

BigInt no Javascript

O BigInt possui algumas observações, sendo elas:

- ▶ Não é possível fazer operações entre um BigInt e um Number.
- ▶ O BigInt não possui casas decimais.

Métodos de number no Javascript: alguns métodos

Método	Descrição
toString()	Retorna um número como uma cadeia de caracteres
toExponential()	Retorna um número escrito em notação exponencial
toFixed()	Retorna um número gravado com um número de decimais
toPrecision()	Retorna um número gravado com um comprimento especificado
ValueOf()	Retorna um número como um número

Métodos de number no Javascript: toFixed()

O método 'toFixed' serve para definir a quantidade de casas decimais que um número inteiro deve ter, por exemplo:

```
let x = 9.656;
document.getElementById("demo").innerHTML =
  x.toFixed(0) + "<br>" +
  x.toFixed(2) + "<br>" +
  x.toFixed(4) + "<br>" +
  x.toFixed(6);
```

10
9.66
9.6560
9.656000

OBS: toFixed(2) é o ideal para trabalhar com dinheiro.

Métodos de number no Javascript: toPrecision()

O método 'toPrecision' é similar ao 'toFixed', a diferença é que ele inclui os números inteiros junto. Nesse caso, ele especifica a quantidade de número ao todo, não só em casas decimais. Por exemplo:

```
let x = 9.656;
document.getElementById("demo").innerHTML =
  x.toPrecision() + "<br>" +
  x.toPrecision(2) + "<br>" +
  x.toPrecision(4) + "<br>" +
  x.toPrecision(6);
```

9.656
9.7
9.656
9.65600

OBS: Em nenhum de seus valores, pode conter o número 0 entre os parênteses

Métodos de conversão: convertendo dados para números

Método	Descrição
Number()	Retorna um número convertido de seu argumento.
parseFloat()	Analisa seu argumento e retorna um número de ponto flutuante
parseInt()	Analisa seu argumento e retorna um número inteiro

Tudo que estiver entre esses parênteses será convertido em um dos 3 tipos de números informados. Ele é extremamente útil caso você vá retirar algum dado do <input>, pois todos os dados inseridos no <input> retornam como 'string'.

Propriedades do number em Javascript:

Propriedade	Descrição
EPSILON	A diferença entre 1 e o menor número > 1.
MAX_VALUE	O maior número possível em JavaScript
MIN_VALUE	O menor número possível em JavaScript
MAX_SAFE_INTEGER	O número inteiro máximo seguro ($2^{53} - 1$)
MIN_SAFE_INTEGER	O inteiro seguro mínimo -($2^{53} - 1$)
POSITIVE_INFINITY	Infinito (retornado no estouro)
NEGATIVE_INFINITY	Infinito negativo (retornado no estouro)
NaN	Um valor "Not-a-Number"

Propriedades do number em Javascript: sintaxe

A sintaxe dessas propriedades é extremamente básica:

Number.NOME_PROPRIEDADE

Normalmente o valor é guardado dentro das variáveis, por exemplo:

```
let x = Number.MAX_VALUE;  
document.getElementById("demo").innerHTML = x; 1.7976931348623157e+308
```

Arrays no Javascript

Um array é basicamente uma variável que pode receber mais de um valor dentro dele, normalmente é lido como uma espécie de lista. Por exemplo:

```
const projetos_fazer = ["HTML", "CSS", "Bootstrap"];
document.getElementById("demo").innerHTML = projetos_fazer;
```

HTML,CSS,Bootstrap

Repare que todos os valores do array foram exibidos. Para mostrar apenas um valor, basta informar o índice entre colchetes[]

OBS: O índice do array começa em 0.

```
projetos_fazer[1]; CSS
```

Arrays no Javascript: Por que usar arrays?

É indispensável o uso dos arrays. Pois eles economizam uma quantidade surreal de memória do computador, pelo simples fato de que cada espaço que uma variável gasta, independente de quanto conteúdo ela possui, é muito grande. Então o array permite um melhor uso para todo esse espaço que fica vago guardando apenas 1 item com mais deles.

Arrays no Javascript: sintaxe

A sintaxe do array é a seguinte:

```
const array_name = [item1, item2, ...];
```

É uma prática comum definir arrays com uma 'const'.

Espaços e quebras de linha não são importantes para o array, então ele também pode ser feito desta forma:

```
const materias = [  
    "Back-End",  
    "Front-End",  
    "Banco de Dados",  
];
```

```
const materias = [];  
materias[0]= "Back-End";  
materias[1]= "Front-End";  
materias[2]= "Banco de Dados";  
document.getElementById("demo").innerHTML = materias;
```

Arrays no Javascript: alterando apenas um item

Para alterar apenas 1 dos itens de um array, basta informá-lo pelo índice e alterar o valor. Desta forma:

```
materias[0] = 'Java';
document.getElementById("demo").innerHTML = materias;
```

Java,Front-End,Banco de Dados

OBS: Um array pode receber dados de diversos tipos diferentes sem problema algum. Assim como é possível ter um array dentro de outro.

Arrays no Javascript: length

O método length retorna a quantidade de elementos que o array possui, se ele tiver 5 elementos, irá retornar o número 5 e por ai vai:

```
document.getElementById("demo").innerHTML = materias.length; 3
```

Inclusive, pode usar esta forma para acessar o último elemento, sem que precise ficar contando quantos elementos possui:

```
document.getElementById("demo").innerHTML = materias[materias.length - 1];  
Banco de Dados
```

Arrays no Javascript: push()

A forma mais fácil de adicionar um novo elemento no javascript sem que precise ficar usando o índice específico, basta usar o método 'push'. Por exemplo:

```
const pessoas = [
  'Karen',
  'Lucas',
  'Alanis',
  'Sarah',
  'Felipe'
];
pessoas.push('Allan');
document.getElementById("demo").innerHTML = pessoas;
```

Karen, Lucas, Alanis, Sarah, Felipe, Allan

Arrays no Javascript: adicionando valores com length

Mas caso você queira adicionar valores sem o push, basta usar o 'nomeArray[nomeArray.length]' para adicionar o valor por último automaticamente. Por exemplo:

```
pessoas[pessoas.length] = 'Isaque';
document.getElementById("demo").innerHTML = pessoas;
```

Karen,Lucas,Alanis,Sarah,Felipe,Allan,Isaque

OBS: No entanto, dessa forma é capaz de gerar algum tipo de erro.

Arrays no Javascript: isArray()

O método 'isArray' verifica se determinada variável ou valor é um array.

Sua sintaxe é:

```
Array.isArray(nomeArray)
```

O valor que será gerado (true ou false) pode ser armazenado dentro de uma variável ou exibido diretamente. Por exemplo:

```
document.getElementById("demo").innerHTML = Array.isArray(pessoas); true
```

Métodos para arrays no Javascript: `toString()`

O método ‘`toString`’ não é exclusivo para arrays e pode ser usado em qualquer tipo de dado, fora string. Ele transforma o array em uma variável do tipo string, onde todos os valores se tornam um só (separados por vírgula). Por exemplo:

```
let caracteres = pessoas.toString()  
document.getElementById("demo").innerHTML = caracteres + '<br>' +  
typeof caracteres;
```

Karen,Lucas,Alanis,Sarah,Felipe,Allan,Isaque
string

Métodos para arrays no Javascript: join()

Este método é similar ao ‘toString’ a diferença é que você pode escolher a forma em que os valores ficarão separados por meio do parâmetro, por exemplo:

```
let caracteres = pessoas.join(' - ')
document.getElementById("demo").innerHTML = caracteres + '<br>' +
typeof caracteres;
```

Karen - Lucas - Alanis - Sarah - Felipe - Allan - Isaque
string

Métodos para arrays no Javascript: push() e pop()

Ao contrário do método 'push', que serve para adicionar um novo valor ao array na última posição, o método 'pop' serve para remover o último valor do array. Por exemplo:

```
const objetos = [
    'teclado',
    'notebook',
    'mouse',
    'monitor',
];
objetos.pop();
document.getElementById("demo").innerHTML = objetos;
```

teclado,notebook,mouse

Métodos para arrays no Javascript: shift()

O método 'shift' também serve para remover os valores como o 'pop', a diferença é que ele remove o primeiro valor do array, e subtrai 1 dos índices de todos os outros valores. Por exemplo:

```
objetos.shift();
document.getElementById("demo").innerHTML = objetos;
```

notebook, mouse, monitor

Caso você use este método e o armazene em uma variável, ela receberá o valor que o 'shift' jogou fora. Por exemplo:

```
let x = objetos.shift();
document.getElementById("demo").innerHTML = x;
```

teclado

Métodos para arrays no Javascript: unshift()

Se o ‘shift’ remove o primeiro valor, o ‘unshift’ serve para adicionar um valor na primeira posição do array, ao fazer isso ele atualiza somando +1 a todos os outros índices. Por exemplo:

```
objetos.unshift('celular');
document.getElementById("demo").innerHTML = objetos;
celular,teclado,notebook,mouse,monitor
```

Métodos para arrays no Javascript: concat()

O método 'concat' também serve para concatenar arrays, especificando o array a ser concatenado como parâmetro dentro dos parênteses, desse jeito:

```
const nomes = [
  'Alvaro',
  'Felipe',
  'Isaque',
  'Gabriel',
  'Gabriel',
  'Lucas',
];
const sobrenomes = [
  'Carvalho',
  'Marcelino',
  'Gonçalves',
  'Gutierrez',
  'Freitas',
  'Bizarria',
];
const novoArray = nomes.concat(sobrenomes);
document.getElementById("demo").innerHTML = novoArray;
```

Alvaro,Felipe,Isaque,Gabriel,Gabriel,Lucas,Carvalho,Marcelino,Gonçalves,Gutierrez,Freitas,Bizarria

Métodos para arrays no Javascript: splice()

Há dois tipos de uso para o ‘splice’, mas vamos por partes. Ele serve tanto para adicionar, quanto para remover elementos. Sua diferença dos anteriores é que ele pode estar removendo no meio do array, através dos índices. Por exemplo:

```
const objetos = ['teclado', 'monitor', 'mouse'];
objetos.splice(1, 0, 'notebook');

document.getElementById("demo").innerHTML = objetos;
```

teclado,notebook,monitor,mouse

O primeiro número informado indica a posição em que ele irá adicionar ou remover. Já o segundo, indica quantos elementos serão removidos e como foi informado 0, nenhum foi.

Métodos para arrays no Javascript: splice()

Agora para remover vamos fazer o seguinte:

```
const objetos = ['teclado', 'monitor', 'mouse',];
objetos.splice(1, 2, 'notebook');
document.getElementById("demo").innerHTML = objetos;
```

teclado,notebook

O segundo número indica quantos valores **a partir** da posição informada serão removidos.

Como visto no exemplo acima, é possível adicionar e remover valores ao mesmo tempo com o ‘splice’.

Ele também pode adicionar mais de um valor de uma vez, basta colocar uma vírgula depois do “notebook” e sair adicionando.

Métodos para arrays no Javascript: Array.from()

O método ‘Array.from’ serve para transformar o parâmetro informado em um array, onde cada caractere é um valor do array. Por exemplo:

```
const myArr = Array.from("ABCDEFG");
document.getElementById("demo").innerHTML = myArr; A,B,C,D,E,F,G
```

Métodos para arrays no Javascript: includes()

O método 'includes' vai verificar se a palavra informada dentro dos parênteses está em algum valor do array, caso sim, ele retorna 'true'. Por exemplo:

```
const materias = ["HTML", "CSS", "Bootstrap", "Javascript"];
document.getElementById("demo").innerHTML = materias.includes("Javascript");
```

true

Organizando arrays no Javascript: sort()

O método sort() serve para organizar o array em ordem alfabética.

Por exemplo:

```
const frutas = ["Banana", "Laranja", "Maçã", "Carambola"];
document.getElementById("demo1").innerHTML = frutas;

frutas.sort();
document.getElementById("demo2").innerHTML = frutas;
```

Banana,Laranja,Maçã,Carambola

Banana,Carambola,Laranja,Maçã

Juntando com o método 'reverse' você consegue organizar o array em ordem alfabética decrescente. Por exemplo:

```
frutas.sort();
frutas.reverse();
document.getElementById("demo2").innerHTML = frutas;
```

Banana,Laranja,Maçã,Carambola

Maçã,Laranja,Carambola,Banana

Organizando arrays numéricas no Javascript: sort()

O 'sort' funciona muito bem para organizar arrays que possuem apenas cadeias de caracteres como valores, mas e se seus valores fossem números ao invés de string? Para isso é só usar a 'função de comparação' como parâmetro. Desse jeito:

```
const pontos = [40, 100, 1, 5, 25, 10];
document.getElementById("demo1").innerHTML = pontos;


pontos.sort(function(a, b){return a - b});
document.getElementById("demo2").innerHTML = pontos;


```

40,100,1,5,25,10

1,5,10,25,40,100

Também é possível fazer ela ficar em ordem decrescente, basta inverter o 'a' com o 'b' no 'return'. Dessa forma:

```
pontos.sort(function(a, b){return b - a});           40,100,1,5,25,10  
document.getElementById("demo2").innerHTML = pontos; 100,40,25,10,5,1
```

Organizando arrays no Javascript: reverse()

Como visto anteriormente, o método 'reverse' serve para inverter a ordem do array. Por exemplo, o último elemento iria virar o primeiro, assim como vice-versa. Dessa forma:

```
const frutas = ["Banana", "Laranja", "Maçã", "Carambola"];
document.getElementById("demo1").innerHTML = frutas;
frutas.reverse();
document.getElementById("demo2").innerHTML = frutas;
```

Banana,Laranja,Maçã,Carambola

Carambola,Maçã,Laranja,Banana

Fazendo iterações nos arrays: map()

Este método serve para criar uma nova função a partir de outras operações. Veja esse código para facilitar mais a compreensão:

```
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);
|
document.getElementById("demo").innerHTML = numbers2;

function myFunction(value) {
    return value * 2;
}
```

90,8,18,32,50

Fazendo iterações nos arrays: map()

```
const numbers2 = numbers1.map(myFunction);
```

Nesta linha podemos ver que uma nova constante é criada para receber o primeiro array, e usa o método 'map()' para enviar o nome de uma função.

```
function myFunction(value) {  
    return value * 2;  
}
```

Nesta função nós dizemos que cada valor será multiplicado por 2 neste novo array.

```
document.getElementById("demo").innerHTML = numbers2;
```

E nesta linha nós exibimos o novo array com as alterações já feitas.

Fazendo iterações nos arrays: filter()

O ‘filter’ serve justamente para fazermos algum tipo de filtro nos arrays.
Por exemplo:

```
const maior18 = numeros.filter(myFunction);

document.getElementById("demo").innerHTML = maior18;

function myFunction(value, index, array) {
  return value > 18;
}
```

No ‘filter’ é possível fazer alguma verificação, nesse caso para retornar os números que são maiores que 18.

45,25

Fazendo iterações nos arrays: reduce()

O ‘reduce’ serve para reduzir todos os valores do array em um só, os somando, subtraindo, multiplicando, etc. Por exemplo:

```
const numeros = [45, 4, 9, 16, 25];
const soma = numeros.reduce(myFunction);

document.getElementById("demo").innerHTML = soma;

function myFunction(total, value, index, array) {
    return total + value;
}
```

99

A função agora irá receber o total como parâmetro também, e o total irá somar 2 valores, acumulá-los e somá-los com o próximo valor.

Fazendo iterações nos arrays: every()

O método 'every' faz uma verificação se todos elementos a cumprem, caso sim, retorna true, do contrário ele retorna false. Por exemplo:

```
const numeros = [45, 4, 9, 16, 25];
let maiores18 = numeros.every(myFunction);

document.getElementById("demo").innerHTML = "Todos são maiores que 18: " +
maiores18;

function myFunction(value, index, array) {
    return value > 18;
}
```

Todos são maiores que 18: **false**

Ele faz a mesma verificação que o 'filter', porém ao invés de retornar os números que são maiores que 18, ele apenas diz se todos são, ou não.

Fazendo iterações nos arrays: find()

O ‘find’ faz uma verificação como o ‘filter’ e o ‘every’, porém ele irá retornar apenas o primeiro valor que cumpre com a verificação:

```
const numbers = [4, 9, 16, 25, 29];
let primeiro = numbers.find(myFunction);

document.getElementById("demo").innerHTML = "O primeiro número maior que 18 é:
" + primeiro;

function myFunction(value, index, array) {
  return value > 18;
}
```

O primeiro número maior que 18 é: 25

Fazendo iterações nos arrays: findIndex()

Se o ‘find’ exibe o primeiro número que cumprir com a verificação, o `findIndex` exibe o índice deste número. Por exemplo no caso anterior:

```
const numbers = [4, 9, 16, 25, 29];

document.getElementById("demo").innerHTML = "O índice do primeiro número maior
que 18 é: " + numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

O índice do primeiro número maior que 18 é: 3

Objetos de data no Javascript

Os objetos de data no javascript nos permite conseguir trabalhar com as datas, tanto acessando ela como fazendo verificações a respeito.
Por exemplo:

```
const d = new Date();
document.getElementById("demo").innerHTML = d;
```

```
Wed Mar 22 2023 19:18:17 GMT-0300 (Horário Padrão de Brasília)
```

OBS: Os objetos de data são “estáticos”, eles não se atualizam a cada segunda, mas há diversos projetos a respeito no youtube.

Criando objetos de data no Javascript

Para criar um objeto de data, basta usar o construtor 'new Date()'.

Existem 9 maneiras de criar um objeto de data, sendo elas:

```
new Date()  
new Date(date string)  
  
new Date(year,month)  
new Date(year,month,day)  
new Date(year,month,day,hours)  
new Date(year,month,day,hours,minutes)  
new Date(year,month,day,hours,minutes,seconds)  
new Date(year,month,day,hours,minutes,seconds,ms)  
  
new Date(milliseconds)
```

OBS: Nestas 8 formas, são quando você coloca algum valor como parâmetro.

Criando objetos de data no Javascript: new Date()

O construtor 'new Date()' cria um objeto de data com a data e hora atuais:

```
const d = new Date();
document.getElementById("demo").innerHTML = d;
Wed Mar 22 2023 19:28:58 GMT-0300 (Horário Padrão de Brasília)
```

Apesar de não atualizar seu valor automaticamente, sempre que recarregar a página ele será editado.

Criando objetos de data no Javascript: new Date(date string)

O ‘new Date(date string)’ serve para exibir a cadeia de caracteres informada em formato de data. Por exemplo:

```
const d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
```

```
Mon Oct 13 2014 11:13:00 GMT-0300 (Horário Padrão de Brasília)
```

Criando objetos de data no Javascript: new Date(ano, mês...)

O construtor 'new Date()' pode receber até 7 valores numéricos como parâmetros, e assim escrevê-los em formato de data. Por exemplo:

```
const d = new Date(2018, 11, 24, 10, 33, 30, 0);  
document.getElementById("demo").innerHTML = d;
```

Mon Dec 24 2018 10:33:30 GMT-0200 (Horário de Verão de Brasília)

OBS: Não se assuste de dezembro ser representado pelo número 11. No javascript os números começam a partir de 0 (nesse caso, ele representa Janeiro).

Além disso, como pode ver, a forma de exibição neste caso é 'mês, dia, ano'.

Criando objetos de data no Javascript: new Date(ano, mês...)

No entanto, colocar um valor maior que 11 não ocasionará em um erro, ele simplesmente irá somar o “adicional” ao ano seguinte. Por exemplo:

```
const d = new Date(2018, 12, 24, 10, 33, 30, 0);
document.getElementById("demo").innerHTML = d;
Thu Jan 24 2019 10:33:30 GMT-0200 (Horário de Verão de Brasília)
```

O mesmo se aplica ao dia, já que escrever um valor maior que o máximo não irá gerar um erro, mas sim será acrescentado ao mês,

```
const d = new Date(2018, 05, 35, 10, 33, 30, 0);
document.getElementById("demo").innerHTML = d;
Thu Jul 05 2018 10:33:30 GMT-0300 (Horário Padrão de Brasília)
```

Criando objetos de data no Javascript: new Date()

A quantidade de números especificam qual parâmetro estão especificando, por exemplo:

6 números especificam respectivamente: ano, mês, dia, hora, minuto e segundo.

```
const d = new Date(2018, 11, 24, 10, 33, 30);
document.getElementById("demo").innerHTML = d;
Mon Dec 24 2018 10:33:30 GMT-0200 (Horário de Verão de Brasília)
```

5 números especificam respectivamente: ano, mês, dia, hora e minuto. (e por assim vai):

```
const d = new Date(2018, 11, 24, 10, 33);
document.getElementById("demo").innerHTML = d;
Mon Dec 24 2018 10:00:00 GMT-0200 (Horário de Verão de Brasília)
```

```
const d = new Date(2018, 11, 24, 10);
document.getElementById("demo").innerHTML = d;
Mon Dec 24 2018 10:00:00 GMT-0200 (Horário de Verão de Brasília)
```

Criando objetos de data no Javascript: new Date()

Não é possível omitir o mês (segundo parâmetro), pois caso você especifique apenas 1, ele será tratado como milissegundos.

Caso você informe apenas 2 números quando for se referir ao ano, eles serão representados como 19xx (x no caso é o número informado). Por exemplo:

```
const d = new Date(20, 11, 24);
document.getElementById("demo").innerHTML = d;
Fri Dec 24 1920 00:00:00 GMT-0300 (Horário Padrão de Brasília)
```

Métodos de data no Javascript:

Quando é criado um objeto de data, ele pode ser exposto a diversos métodos especificados para datas permitindo maior interação com elas.

Por exemplo o método `toDateString()`, que permite que a exibição da data se torne mais legível. Desta forma:

```
const d = new Date();
```

```
document.getElementById("demo").innerHTML = d.toDateString();
```

Wed Mar 22 2023

Métodos de datas no Javascript: toUTCString()

O método 'toUTCString()' faz com que a data seja convertida em uma cadeia de caracteres do tipo string. Também é bom por melhorar a legibilidade. Por exemplo:

Usando o

```
const d = new Date();
document.getElementById("demo").innerHTML = d.toUTCString();
```

Wed, 22 Mar 2023 23:42:07 GMT

usando:

```
const d = new Date();
document.getElementById("demo").innerHTML = d;

```

Wed Mar 22 2023 20:43:48 GMT-0300 (Horário Padrão de Brasília)

Não

Formatos de datas no Javascript

Existem 3 tipos de formatos de datas, sendo eles:

Tipo	Exemplo
Data ISO	"2015-03-25" (O Padrão Internacional)
Data Abreviada	"03/25/2015"
Data longa	"Mar 25 2015" ou "25 Mar 2015"

Estes formatos são especificados através do 'new Date(date string)'.

```
new Date("2015-03-25");
```

Formatos de datas no Javascript: ISO

O formato de data 'ISO' é usado de forma padrão no javascript, nele os números são separados entre traços (-). Por exemplo:

```
const d = new Date("2015-03-25");
Tue Mar 24 2015 21:00:00 GMT-0300 (Horário Padrão de Brasília)
```

OBS: Pode acabar aparecendo dia 24 devido ao fuso horário. Além disso, quando o valor passado é através de uma cadeia de caracteres, o mês volta a começar a partir do 1 (representando Janeiro).

Formatos de datas no Javascript: ISO

Neste formato de data, é possível estar escrevendo sem especificar o mês. Neste caso, o resultado será o último dia do ano anterior ao informado. Por exemplo:

Caso você informe o ano 2020, será exibido algo em torno de:

“Dec, 31, 2019”.

```
const d = new Date("2015");
document.getElementById("demo").innerHTML = d;
Wed Dec 31 2014 22:00:00 GMT-0200 (Horário de Verão de Brasília)
```

Formatos de data no Javascript: ISO

Caso você queira informar a data e hora, basta separá-los por uma letra T (maiúscula).

Além disso, a hora UTC (Tempo Universal Coordenado) é definida usando uma letra Z (maiúscula).

Por exemplo:

```
const d = new Date("2015-03-25T12:00:00Z");
```

A letra T servirá para separar a data da hora.

Formatos de datas no Javascript: fuso horário

Caso você crie um objeto de data sem definir o fuso horário, ele será definido a partir do fuso horário do navegador. Por exemplo, caso você defina uma data e hora, ela será exibida de forma diferente para uma pessoa que mora em outro país.

Por isso deve tomar bastante cuidado e sempre definir o fuso horário

Formatos de datas no Javascript: datas curtas

As “datas curtas” são definidas com a sintaxe “MM/DD/AAAA”. Por exemplo:

```
const d = new Date("03/25/2015");
```

Um aviso, o uso do 0 é obrigatório, uma vez que sua ausência pode acabar ocasionando em erros.

Uma “data curta” definida como “AAAA/MM/DD” poderá retornar “undefined”. Por isso deve sempre declará-la como na sintaxe acima.

Formatos de datas no Javascript: datas longas

Datas longas são frequentemente escritas com a sintaxe:
“MMM DD AAAA”. Por exemplo:

```
const d = new Date("Mar 25 2015");
```

Diferente das “datas curtas”, as datas longas podem informar o mês e o dia em qualquer ordem. Por exemplo:

```
const d = new Date("25 Mar 2015");
```

Formatos de datas no Javascript: datas longas

Além disso, o nome do mês pode ser escrito de forma completa, ou de forma abreviada:

```
const d = new Date("January 25 2015");
```

Outra forma de definir uma data longa é separando os valores por vírgula, uma vez que elas são ignoradas. Além disso, ele não diferencia letras maiúsculas de minúsculas:

```
const d = new Date("JANUARY, 25, 2015");
```

Métodos para obter datas no Javascript: lista de métodos

Método	Descrição
getFullYear()	Obter ano como um número de quatro dígitos (aaaa)
getMonth()	Obter mês como um número (0-11)
getDate()	Obter dia como um número (1-31)
getDay()	Obter dia da semana como um número (0-6)
getHours()	Obter hora (0-23)
getMinutes()	Obter minuto (0-59)
getSeconds()	Obter o segundo (0-59)
getMilliseconds()	Obter milissegundos (0-999)
getTime()	Obter tempo (milissegundos desde 1º de janeiro de 1970)

Métodos para obter datas no Javascript: getFullYear()

O método 'getFullYear()' serve para retornar apenas o ano do valor especificado e o exibe com 4 dígitos. Por exemplo:

```
const d = new Date()  
document.getElementById("demo").innerHTML = d.getFullYear();
```



Métodos para obter datas no Javascript: getMonth

Este método retorna o mês de uma data com um número de 0 a 11.
Por exemplo:

```
const d = new Date("November 23 2018")
document.getElementById("demo").innerHTML = d.getMonth(); 10
```

Lembrando que no javascript, janeiro é o mês número 0, fevereiro é o mês número 1 e por assim vai.

Métodos para obter datas no Javascript: getDate()

Este método retorna o dia de uma data, podendo retornar um número de 1 até 31. Por exemplo:

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getDate();
```

22

Métodos para obter datas no Javascript: getHours()

Este método retorna o valor da hora informada ou que foi buscada no construtor 'new Date()'. Ele pode retornar um número de 0 até 23. Por exemplo:

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getHours();
```

21

Métodos para obter datas no Javascript: getMinutes()

Este método serve para retornar os minutos informados ou que foram recolhidos do construtor 'new Date()'. Ele pode retornar um número de 0 até 59. Por exemplo:

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getMinutes();
```

38

Métodos para obter datas no Javascript: getSeconds()

Este método serve para retornar os segundos informados ou recolhidos. Ele pode retornar um número de 0 até 59. Por exemplo:

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getSeconds();
```

40

Métodos para obter datas no Javascript: getMilliseconds()

Este método serve para retornar os milissegundos informados ou recolhidos. Ele pode retornar de um número de 0 até 999. Por exemplo:

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getMilliseconds(); 960
```

Métodos para obter datas no Javascript: getDay()

Este método retorna o dia da semana em um número de 0 até 6, onde domingo vale 0, segunda vale 1 e por assim vai. Por exemplo:

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getDay(); 3
```

OBS: Se apareceu 3, significa que hoje é quarta.

Métodos para obter datas no Javascript: getTime()

Este método retorna os milissegundos atuais desde 1 de janeiro de 1970.

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getTime(); 1679532526165
```

OBS: Ele também pode ser usado caso informe alguma data específica. Por exemplo:

```
const d = new Date("February 15 2001");
document.getElementById("demo").innerHTML = d.getTime(); 982202400000
```

Métodos para obter datas no Javascript: getTimezoneOffset

Este método retorna a diferença (em minutos) entre a hora local e a hora UTC. Por exemplo:

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getTimezoneOffset();
```

180

Métodos para definir datas no Javascript: lista

Método	Descrição
setDate()	Definir o dia como um número (1-31)
setFullYear()	Definir o ano (opcionalmente mês e dia)
setHours()	Definir a hora (0-23)
setMilliseconds()	Definir os milissegundos (0-999)
setMinutes()	Definir os minutos (0-59)
setMonth()	Definir o mês (0-11)
setSeconds()	Definir os segundos (0-59)
setTime()	Definir o tempo (milissegundos desde 1º de janeiro de 1970)

Métodos para definir datas no Javascript: setFullYear()

Este método serve para definir um ano, você pode recolher seu dia atual pelo construtor 'new Date()' e depois redefinir o ano usando o 'setFullYear'. Por exemplo:

```
const d = new Date();
d.setFullYear(2020);
document.getElementById("demo").innerHTML = d;
Sun Mar 22 2020 23:30:54 GMT-0300 (Horário Padrão de Brasília)
```

Ele também pode **opcionalmente** definir o mês e dia. Por exemplo:

```
const d = new Date();
d.setFullYear(2020, 11, 4);
document.getElementById("demo").innerHTML = d;
Fri Dec 04 2020 23:32:17 GMT-0300 (Horário Padrão de Brasília)
```

Métodos para definir datas no Javascript: setMonth()

Este método serve para definir o mês através de valores que vão do 0 até o 11. Por exemplo:

```
const d = new Date();
d.setMonth(9);
document.getElementById("demo").innerHTML = d;
Sun Oct 22 2023 23:34:23 GMT-0300 (Horário Padrão de Brasília)
```

Métodos para definir datas no Javascript: setDate()

Este método serve para definir o dia, pode receber valores de 1 até 31. Por exemplo:

```
const d = new Date();
d.setDate(15);
document.getElementById("demo").innerHTML = d;
Wed Mar 15 2023 23:46:59 GMT-0300 (Horário Padrão de Brasília)
```

Com ele você também pode **adicionar** dias, caso eles passem do mês, serão acrescentados no próximo. Por exemplo:

```
const d = new Date();
d.setDate(d.getDate() + 17);
document.getElementById("demo").innerHTML = d;
Sat Apr 08 2023 23:48:58 GMT-0300 (Horário Padrão de Brasília)
```

Métodos para definir datas no Javascript: setHours()

Este método serve para definir a hora, pode receber valores de 0 até 23. Por exemplo:

```
const d = new Date();
d.setHours(15);
document.getElementById("demo").innerHTML = d;
Wed Mar 22 2023 15:50:22 GMT-0300 (Horário Padrão de Brasília)
```

Métodos para definir datas no Javascript: setMinutes()

Este método serve para definir os minutos que foram recolhidos. Pode receber valores de 0 até 59, por exemplo:

```
const d = new Date();
d.setMinutes(45);
document.getElementById("demo").innerHTML = d;
Wed Mar 22 2023 23:45:08 GMT-0300 (Horário Padrão de Brasília)
```

Métodos para definir datas no Javascript: setSeconds()

Este método serve para definir os segundos que foram recolhidos. Pode receber valores de 0 até 59, por exemplo:

```
const d = new Date();
d.setSeconds(37);
document.getElementById("demo").innerHTML = d;
Wed Mar 22 2023 23:53:37 GMT-0300 (Horário Padrão de Brasília)
```

Constantes matemáticas no Javascript:

O javascript fornece algumas constantes que podem ser acessadas como propriedades matemáticas. Sendo elas:

<code>Math.E</code>	retorna o número de Euler
<code>Math.PI</code>	retorna PI
<code>Math.SQRT2</code>	retorna a raiz quadrada de 2
<code>Math.SQRT1_2</code>	retorna a raiz quadrada de 1/2
<code>Math.LN2</code>	retorna o logaritmo natural de 2
<code>Math.LN10</code>	retorna o logaritmo natural de 10
<code>Math.LOG2E</code>	retorna o logaritmo de base 2 de E
<code>Math.LOG10E</code>	retorna o logaritmo de base 10 de E

OBS: Existem algumas outras que também serão informadas.

Constantes matemáticas no Javascript:

A sintaxe dessas constantes é bem básica, sendo:

`Math.metodo(numero)`

Há algumas outras constantes de matemática, que são:

<code>Math.round(x)</code>	Retorna x arredondado para o inteiro mais próximo
<code>Math.ceil(x)</code>	Retorna x arredondado para cima para o inteiro mais próximo
<code>Math.floor(x)</code>	Retorna x arredondado por defeito para o inteiro mais próximo
<code>Math.trunc(x)</code>	Retorna a parte inteira de x (new in ES6)

O ‘`Math.floor`’ o arredonda para baixo.

Constantes matemáticas no Javascript: Math.round()

O 'Math.round' serve para arredondar um número que possui casas decimais para seu inteiro mais próximo. Por exemplo, caso o valor das casas decimais seja igual ou maior que 5, o número será arredondado para uma casa acima. Mas caso seja inferior a 5, ele irá retirar as casas decimais do número. Por exemplo:

```
document.getElementById("demo").innerHTML = Math.round(4.6); 5
```

```
document.getElementById("demo").innerHTML = Math.round(4.5); 5
```

```
document.getElementById("demo").innerHTML = Math.round(4.4); 4
```

Constantes matemáticas no Javascript: Math.ceil()

A constante 'Math.ceil' também arredonda um número decimal para inteiro, a diferença é que não importa o valor da casa decimal, ele sempre irá arredondar o número para cima. Por exemplo:

```
document.getElementById("demo").innerHTML = Math.ceil(4.6); 5
```

```
document.getElementById("demo").innerHTML = Math.ceil(4.4); 5
```

Constantes matemáticas no Javascript: Math.floor()

Ao contrário da constante anterior, o ‘Math.floor’ irá sempre arredondar um número real para baixo, ou seja, usar esta constante é o mesmo que arredondar um número sempre para sua versão inteira, não importando o quanto alto é o valor das casas decimais. Por exemplo:

```
Math.floor(4.4); 4
```

```
Math.floor(4.9); 4
```

Constantes matemáticas no Javascript: Math.trunc()

Se as constantes anteriores serviam para arredondar um número real para seu inteiro mais próximo com base no valor das casas decimais. A constante 'Math.trunc' apenas descarta as casas decimais do número, retornando apenas seu valor inteiro. Por exemplo:

```
Math.trunc(4.7); 4
```

```
Math.trunc(-4.2); -4
```

Constantes matemáticas no Javascript: Math.sign()

A constante 'Math.sign()' pode retornar 3 valores, sendo '1 (caso um valor positivo seja colocado entre parênteses)', '0 (caso o valor enviado seja nulo ou 0)' e '-1 (caso o valor enviado seja negativo)'.

Por exemplo:

document.write(Math.sign(4.2) + ' ');	1
document.write(Math.sign(0) + ' ');	0
document.write(Math.sign(-4.2) + ' ');	-1

Constantes matemáticas no Javascript: Math.pow()

A constante 'Math.pow' recebe dois valores em seus parênteses, o primeiro ele usa como base e o segundo ele o transforma em um expoente, realizando assim a operação. Por exemplo:

```
Math.pow(8,2); 64
```

$$8^2 = 8 * 8 = 64.$$

Constantes matemáticas no Javascript: Math.sqrt()

A constante 'Math.sqrt' serve para retornar o valor da raiz quadrada do número informado. Por exemplo:

```
Math.sqrt(64); 8
```

$$8 * 8 = 64.$$

Constantes matemáticas no Javascript: Math.abs()

Esta constante sempre retorna o valor absoluto do número informado. Para os desavisados, valor absoluto nada mais é do que o próprio número informado, só que **positivo**. Por exemplo:

```
document.write( Math.abs(-3.674) + '<br>' ); 3.674  
document.write( Math.abs(3.674) ); 3.674
```

OBS: Todas constantes apresentadas até agora servem tanto para números reais como inteiros.

Constantes matemáticas no Javascript: Math.min()

As constantes 'Math.min' e 'Math.max' são usados para descobrir o menor e maior valor colocados entre os parênteses. Por exemplo:

```
Math.min(0, 150, 30, 20, -8, -200) + '<br>' + -200  
Math.max(0, 150, 30, 20, -8, -200); 150
```

Constantes matemáticas no Javascript: Math.random()

O 'Math.random' serve para gerar um número aleatório entre 0 e 1.
Por exemplo:

```
Math.random() 0.00446573028525421
```

Além disso, podemos combiná-lo com um dos 3 'Math' que servem para arredondamento (round, ceil, floor). Nesse caso ele irá gerar 0 ou 1 aleatoriamente caso coloque o 'round'.

```
Math.round(Math.random())
```

Caso queira deixá-lo gerar um número inteiro aleatório de 0 a 9, apenas multiplique o 'Math.random' por 10. Por exemplo:

```
Math.round(Math.random() * 10)
```

Constantes matemáticas no Javascript: Math.random()

Agora vou mostrar algumas formas de exibir diversos tipos de números aleatórios:

```
Math.floor(Math.random() * 11); // para ir de 0 até 10.  
Math.floor(Math.random() * 100); // Para ir de 0 até 99  
Math.floor(Math.random() * 101); // Para ir de 0 até 100.  
Math.floor(Math.random() * 10) + 1; // Para ir de 1 até 10.
```

Agora que viu que possui um padrão, pode ir brincando e alterando os valores de mínimo e máximo do random.

Constantes matemáticas no Javascript: Math.random()

Caso você queira definir um valor mínimo e máximo personalizado.
Basta construir esta função para tal.

```
function getRndInteger(min, max) {  
    return Math.floor(Math.random() * (max - min)) + min;  
}
```

A função irá arredondar o número aleatório para um inteiro, logo
após, ele irá subtrair o valor máximo pelo mínimo e então multiplicar o
resultado pelo número aleatório. Em seguida, ele será arredondado e
somado pelo valor mínimo.

Constantes matemáticas no Javascript: Math.log2()

A constante ‘Math.log2’ serve para vermos em quantas vezes teremos que ‘multiplicar por 2’ até chegar em determinado número. Por exemplo:

```
Math.log2(8); 3
```

Precisa “multiplicar por 2” 3 vezes para chegar a 8. Por exemplo:

$$2 * 2 * 2 = 8.$$

Constantes matemáticas no Javascript: Math.log10()

Essa constante faz a mesma coisa que a anterior, a diferença é que ela multiplica por 10 ao invés de 2.

```
Math.log10(10000); 4
```

$$10 * 10 * 10 * 10 = 10.000$$

Boolean no Javascript

O tipo de dado ‘Boolean’ normalmente serve para fazermos uma verificação se algo é verdadeiro ou falso, servindo perfeitamente para pedirmos ao computador analisar determinadas situações e respostas do usuário e interagir de acordo.

É possível usá-lo quando você armazena o determinado valor (true ou false) em uma variável, ou quando você efetua uma comparação.

Por exemplo:

(10 > 9)

Acredito que já passou pelos operadores de comparação, sendo assim, deve estar ao menos um pouco familiarizado com o ‘boolean’. Dê uma revisada nas páginas para reforçar.

If no Javascript:

Para pedirmos ao navegador verificar alguma coisa, utilizamos a palavra 'if', pois ele realiza uma verificação de 'se'. Por exemplo:
"Se o jogo estiver aberto, envie a mensagem 'jailson a caminho' ".

Sua sintaxe é:

```
if (condição) {  
    código  
}
```

Lembre-se que deve escrever tudo em letras minúsculas.

OBS: Nesse caso, se a condição for falsa, o computador ignora o código dentro do 'if'.

If no Javascript: prática

Vamos testar o seguinte exemplo:

Caso o 'if' seja verdadeiro:

```
let idade = 18;  
  
if (idade >= 18) {  
    document.write('Já é maior de idade');  
}
```

Já é maior de idade

Caso o 'if'

```
let idade = 17;  
  
if (idade >= 18) {  
    document.write('Já é maior de idade');  
}
```



Como pode ver, ele ignora completamente o 'if' caso ele seja falso. Então no próximo slide vamos ver outra palavra para usar caso o 'if' não seja verdadeiro.

If e else no Javascript:

O 'else' serve no caso de ser o 'se não'. Ou seja:

Se (condição for verdadeira) {

 execute o código

} Se não {

 execute este código.

}

Essa também é a sintaxe do if e do else. Lembre-se de não escrevê-los em letras maiúsculas.

O objetivo do else é apresentar outro resultado ao invés do navegador simplesmente ignorar o if, veja na prática:

If e else no Javascript: prática

Caso o 'if' seja verdadeiro:
seja falso:

```
let idade = 18;

if (idade >= 18) {
    document.write('Já é maior de idade');
} else {
    document.write('Não é maior de idade');
}
```

Já é maior de idade

Caso o 'if'

```
let idade = 17;

if (idade >= 18) {
    document.write('Já é maior de idade');
} else {
    document.write('Não é maior de idade');
}
```

Não é maior de idade

Como pode ver, caso o 'se' seja 'falso', o navegador imediatamente cai na declaração 'se não' e executa o código sem verificar qualquer condição.

Else if no Javascript:

Agora você já aprendeu a pedir para o navegador verificar uma determinada condição, caso ela seja verdadeira, pode realizar um código específico. Mas, e se você precisar de mais de 1 condição?

Para isso existe o else if, que é a combinação dos dois, sua sintaxe é a seguinte:

```
if (condição) {  
    Executa o código  
} else if (condição) {  
    Executa o código caso o primeiro seja falso.  
} else {  
    Executa o código caso todas condições forem falsas.  
}
```

Else if no Javascript: prática

```
if (projetos_feitos > 15 && projetos_feitos <= 26) {
    document.write('Parabéns, você está conseguindo concluir os projetos obrigatórios.');

} else if (projetos_feitos > 26 && projetos_feitos <= 43) {
    document.write('Meus mais sinceros parabéns, conseguiu terminar todos os projetos
obrigatórios e agora está fazendo os opcionais, tenha orgulho de si mesmo(a)');

} else {
    document.write('Bora fazer mais projeto, braço curto');
}
```

Parabéns, você está conseguindo concluir os projetos obrigatórios.

```
let projetos_feitos = 10;

if (projetos_feitos > 15 && projetos_feitos <= 26) {
    document.write('Parabéns, você está conseguindo concluir os projetos obrigatórios.');

} else if (projetos_feitos > 26 && projetos_feitos <= 43) {
    document.write('Meus mais sinceros parabéns, conseguiu terminar todos os projetos
obrigatórios e agora está fazendo os opcionais, tenha orgulho de si mesmo(a)');

} else {
    document.write('Bora fazer mais projeto, braço curto');
}
```

Bora fazer mais projeto, braço curto

```
let projetos_feitos = 33;

if (projetos_feitos > 15 && projetos_feitos <= 26) {
    document.write('Parabéns, você está conseguindo concluir os projetos obrigatórios.');

} else if (projetos_feitos > 26 && projetos_feitos <= 43) {
    document.write('Meus mais sinceros parabéns, conseguiu terminar todos os projetos
obrigatórios e agora está fazendo os opcionais, tenha orgulho de si mesmo(a)');

} else {
    document.write('Bora fazer mais projeto, braço curto');
}
```

Meus mais sinceros parabéns, conseguiu terminar todos os projetos
obrigatórios e agora está fazendo os opcionais, tenha orgulho de si mesmo(a)

Operador ternário no Javascript

Agora que vocês aprenderam sobre o 'if', notaram que o código pode acabar ficando bem grande algumas vezes. Caso você queira, é possível escrever um 'desvio condicional' (é o if) de forma mais simples, que é utilizando o operador ternário. Sua sintaxe é:

nomeVariavel = (condição) ? seForVerdadeiro : seForFalso

repare que o operador ternário é uma versão mais simples do 'if e else(se e senão)'.

Vamos utilizar um exemplo de código para facilitar a compreensão:

```
let projetos_feitos = 25;  
  
projetos_feitos = projetos_feitos >= 20 ? document.write('Mandou Bem') : document.write('Precisa  
fazer mais projetos');
```

Mandou Bem

```
let projetos_feitos = 19;  
  
projetos_feitos = projetos_feitos >= 20 ? document.write('Mandou Bem') : document.write('Precisa  
fazer mais projetos');
```

Precisa fazer mais projetos

Switch no Javascript

O ‘switch’ também é um desvio condicional, porém ele não consegue realizar verificações mais aprofundadas. Entretanto, ele é muito mais rápido em comparação ao ‘if’..

Sua sintaxe é a seguinte:

```
let dia = 4;
switch (dia) {
  case 0:
    dia = "Domingo";
    break;
  case 1:
    dia = "Segunda";
    break;
  case 2:
    dia = "Terça";
    break;
  case 3:
    dia = "Quarta";
    break;
  case 4:
    dia = "Quinta";
    break;
  case 5:
    dia = "Sexta";
    break;
  case 6:
    dia = "Sábado";
}
document.getElementById("demo").innerHTML = "Hoje é: " + dia;
```

Switch no Javascript: sintaxe

No começo da sintaxe, podemos ver a palavra 'switch' completamente em letras **minúsculas**. Seguido por parênteses, com o nome da variável que possui, ou que irá receber o valor, no caso é o 'dia'. Após isso é aberto 1 chave com fechamento {}, simbolizando um bloco de código, onde todo código do switch deverá ficar dentro destas chaves ({}).

Switch no Javascript: case

O ‘case’ simboliza uma opção de tecla do usuário, por exemplo se o usuário aperta o número 3, vai exibir quarta (obviamente o código não usou entrada de dados, mas é um exemplo). O ‘case’ não precisa necessariamente ser um número, já que ele simboliza uma tecla ou palavra.

Para facilitar a compreensão, pense em um menu de opções, onde cada ‘case’ é um item do cardápio que o usuário pode informar ao apertar as teclas de 0 até 6 (usando o código anterior como exemplo).

Switch no Javascript: break

Caso você tenha observado a imagem do código, deve ter notado a existência da palavra 'break' que é comum dentro do 'switch'. Isto se deve ao fato de que ela encerra uma opção, em resumo, um 'case' termina quando o navegador vê a palavra 'break'. Caso ele não a encontre, iria percorrer **todas as opções** até achar a palavra 'break' dentro do 'switch'.

Então não se esqueça de sempre colocar ela para encerrar os 'case'.

Switch no Javascript: default

Embora ele não tenha aparecido naquela imagem. O 'default' é o 'else' do 'switch'. Servindo para executar um código caso o usuário faça algo que não está entre os 'case'. Por exemplo:

```
let dia = 9;
switch (dia) {
  case 0:
    dia = "Domingo";
    break;
  case 1:
    dia = "Segunda";
    break;
  case 2:
    dia = "Terça";
    break;
  case 3:
    dia = "Quarta";
    break;
  case 4:
    dia = "Quinta";
    break;
  case 5:
    dia = "Sexta";
    break;
  case 6:
    dia = "Sábado";
    break;
  default:
    document.write('Por favor informe um dia válido');
}
```

```
let dia = 6;
Hoje é: Sábado
```

```
let dia = 9;
Por favor informe um dia válido
```

Switch no Javascript: detalhes

- ▶ Se houver mais de um 'case' com a mesma verificação, o primeiro 'case' será escolhido.
- ▶ Se não houver nenhum 'case' correspondente, o navegador continuará com a leitura padrão do código e irá ignorar o switch.
- ▶ Se o 'break' não for encontrado, o navegador irá ativar o código de todas opções até achar o 'break' ou até o switch acabar.

Loops no Javascript

Os loops podem executar um bloco de código várias vezes.

Por exemplo, suponhamos que você queira colocar a soma dos números de 1 ao 10, mas não quer escrever linha por linha para exibir o valor:

```
let acumuladora = 0;
for (let contadora = 1; contadora <= 10; contadora++) {
    acumuladora = acumuladora + contadora;
    document.write(acumuladora + '<br>');
}
```

Loops no Javascript: while

Existem 4 principais tipos de loop no javascript, sendo 'while', 'do...', 'while', 'for' e 'foreach(para arrays)'.

O primeiro que será aprendido é o 'while':

O loop 'while' é conhecido por fazer a verificação primeiro, antes de executar o código que está dentro dele. Sua sintaxe é:

```
while (condição) {  
    execute o código  
}
```

Caso a condição seja falsa, o navegador ignora o que estiver dentro do 'while'. Mas caso seja verdadeira, ele executa o código que está dentro.

Loops no Javascript: variável contadora

Quando estiver usando um loop, na maioria das vezes, será **primordial** a existência de uma variável contadora, que terá seu valor incrementado a cada vez que o loop resetar. Ela é absolutamente necessária para impedir que o laço de repetição (while, etc) entre em um loop infinito, que não tem como acabar. Por exemplo: Vamos escrever os números de 1 ao 10 usando o while:

Caso não incrementarmos a variável contadora, ela terá sempre o valor 1, logo, o loop nunca iria acabar.

```
let contadora = 1;  
  
while (contadora <= 10) {  
    document.write(contadora + '<br>');  
    contadora++;  
}
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
let contadora = 1;  
  
while (contadora <= 10) {  
    document.write(contadora + '<br>');  
}
```

Loops no Javascript: do... while

Ao contrário do loop 'while', o 'do {} while' fará a verificação **embaixo**, quando o código do loop já tiver acabado. Ele serve para rodar o código pelo menos 1 vez, mesmo que a condição seja falsa. Mas caso a condição seja verdadeira, ele executa o código até ela se tornar falsa. Por exemplo:

```
let contadora = 1;
do {
    document.write(contadora + '<br>');
    contadora--;
} while (contadora >= 10);
```

1

```
let contadora = 20;
do {
    document.write(contadora + '<br>');
    contadora--;
} while (contadora >= 10);
```

20
19
18
17
16
15
14
13
12
11
10

Como a condição é que a contadora seja maior que 10 (é falsa, pois a contadora é 1), o código exibiu a contadora apenas 1 vez. Já que ele executa o código para só depois verificar.

Loops no Javascript: for

Quando o loop ‘for’ é escrito no código, o navegador é obrigado a executá-lo, justamente porque o for não utiliza uma verificação como os outros loops anteriores, mas sim uma verificação para **até quando** ele irá se repetir. Digo isto pelo fato do ‘for’ possuir **começo e fim pré-definidos**. Veja esta imagem para facilitar a compreensão:

```
for (let contadora = 1; contadora <= 10; contadora++) {  
    document.write(contadora + '<br>');  
}
```

Como pode ver, dentro dos parênteses, é declarado a variável contadora, definido até onde ela vai, e no final é dito que ela será incrementada

Loops no javascript: expressões

O ‘for’ é conhecido por ter começo e fim pré-definidos. Além disso, deve ter notado que a ‘condição’ dele é separada por ponto e vírgula (;).

```
(let contadora = 1; contadora <= 10; contadora++)
```

Essas partes separadas são chamadas de expressões, será explicado por partes.

Loops no Javascript: expressão 1

Normalmente a expressão 1 é feita para declarar uma variável que será a contadora, por exemplo:

```
let i = 0;
```

Pelo 'for' possuir o escopo de bloco, as variáveis que são declaradas com o 'let' dentro dele só podem ser usadas dentro do mesmo.

Loops no Javascript: Expressão 2

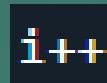
Muitas vezes, a expressão dois será avaliada para definir ‘até onde a expressão 1 vai’. Em outras palavras, ela define o número de repetições que o ‘for’ fará:

```
i < 10;
```

Nesse caso, está dizendo para repetir o laço enquanto ‘i’ for menor que 10.

Loops no Javascript: Expressão 3

A última expressão é usada para definir se a variável definida na expressão 1 será incrementada ou decrementada.



Graças a essas 3 expressões, o código executado dentro do 'for' pode ser apenas aquilo que queremos passar, sem necessidade de se preocupar com incremento ou coisas do tipo, uma vez que todas condições são escritas no começo.

Loops no Javascript: For in

Este loop serve apenas para objetos. Sua sintaxe é bem simples:

```
const pessoa = {  
    primeiroNome:"Fulano",  
    segundoNome:"Beltranado",  
    idade:25,  
};  
  
let txt = "";  
for (let x in pessoa) {  
    txt += pessoa[x] + " ";  
}  
  
document.getElementById("demo").innerHTML = txt;
```

A 'const' pessoa = {} define um objeto. Já o for(let x (serve como contadora) **in** nomeObjeto) irá configurar um loop em que o valor da contadora será incrementado automaticamente. Além disso, eles armazenam os **valores** (não as propriedades) do objeto dentro da variável 'txt', no final o código só exibe.

Break no Javascript:

O ‘break’ pode ser usado fora do ‘switch’, onde é muito útil também nos laços de repetição. Dentro deles o ‘break’ serve justamente para encerrar o loop caso atinja tal determinada condição. Por exemplo:

```
for (let contadora = 1; contadora <= 10; contadora++) {  
    if (contadora == 4) {  
        break;  
    }  
    document.write(contadora + '<br>');  
}
```

```
for (let contadora = 1; contadora <= 10; contadora++) {  
    document.write(contadora + '<br>');  
    if (contadora == 4) {  
        break;  
    }  
}
```

No caso ele parou no ‘3’ porque o incremento está no começo do ‘for’, além disso, o document.write() está embaixo da verificação, dessa forma, ele interrompe o loop antes de exibir o valor.

Continue no Javascript:

Diferente do ‘break’ que encerra o loop quando a verificação de uma condição é verdadeira. O ‘continue’ “pula” essa condição, enquanto o loop funciona normalmente. Por exemplo:

```
for (let contadora = 1; contadora <= 10; contadora++) {  
    if (contadora == 4) {  
        continue;  
    }  
    document.write(contadora + '<br>');  
}
```

1
2
3
5
6
7
8
9
10

Neste caso, é obrigatório o ‘continue’ estar abaixo do `document.write()`.

Conjuntos no Javascript

Um conjunto no Javascript é uma coleção de valores exclusivos, onde cada valor só pode ser armazenado uma única vez, sem repetição.

Alguns métodos essenciais de conjunto:

<code>new Set()</code>	Cria um novo conjunto
<code>add()</code>	Adiciona um novo elemento ao conjunto
<code>delete()</code>	Remove um elemento de um conjunto
<code>has()</code>	Retorna true se um valor existir no Conjunto
<code>forEach()</code>	Invoca um retorno de chamada para cada elemento no Set
<code>values()</code>	Retorna um iterador com todos os valores em um Conjunto

Conjuntos no Javascript: criando um conjunto

Para usar um conjunto, basta transformar um array no mesmo através do código 'new Set()', com os valores do array dentro. Por exemplo:

```
const letras = new Set(["a", "b", "c"]);
```

Para adicionar valores, use o método 'add()' cuja sintaxe é a mesma dos métodos anteriores:

```
const letras = new Set(["a", "b", "c"]);
letras.add("d");
letras.add("e");
letras.add("f");
letras.add("g");
```

Conjuntos no Javascript: variáveis

Com o conjunto também é possível armazenar o valor de variáveis diretamente. Há algumas formas de fazer isso, porém a mais recomendada é através do método 'add()'. Por exemplo:

```
const letras = new Set([]);

const a = 'a';
const b = 'b';
const c = 'c';

letras.add(a);
letras.add(b);
letras.add(c);
```

OBS: Caso você adicione elementos iguais, somente o primeiro adicionado será salvo.

Map no Javascript:

Um mapa é formado por pares de 'índice-valor' onde os índices podem ser qualquer tipo de dado.

Para criar um mapa, deve transformar um array em um da mesma forma que o conjunto. No entanto, cada valor do mapa será coberto de colchetes. Por exemplo:

```
const frutas = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
```

Map no Javascript: alguns métodos essenciais

Método	Descrição
<code>new Map()</code>	Cria um novo mapa
<code>set()</code>	Define o valor de uma chave em um Mapa
<code>get()</code>	Obtém o valor de uma chave em um Mapa
<code>delete()</code>	Remove um elemento Map especificado pela chave
<code>has()</code>	Retorna true se uma chave existir em um Mapa
<code>forEach()</code>	Chama uma função para cada par chave/valor em um Mapa
<code>entries()</code>	Retorna um iterador com os pares [chave, valor] em um Mapa
Propriedade	Descrição
<code>size</code>	Retorna o número de elementos em um Mapa

Map no Javascript: adicionando dados com set()

É possível estar adicionando mais dados fora do 'Map' com o método 'set()', da seguinte forma:

```
const frutas = new Map();
frutas.set('bananas', 500);
frutas.set('maçãs', 300);
frutas.set('uvas', 200);

document.getElementById("demo").innerHTML = frutas.get("maçãs");
```

300

Repare que o método 'get' exibe os **valores** através do **índice** informado.

Map no Javascript: alterando valores com set()

Além de adicionar valores, o 'set' também pode sobrescrevê-los, através do 'índice' do valor.

```
const frutas = new Map([
  ["maçãs", 500],
  ["bananas", 300],
  ["laranjas", 200]
]);

frutas.set("maçãs", 200);

document.getElementById("demo").innerHTML = frutas.get("maçãs");
```

Map no Javascript: retornando valores com get()

O método 'get()' consegue estar retornando um valor ao ter seu índice informado entre parênteses, por exemplo:

```
document.getElementById("demo").innerHTML = frutas.get("maçãs"); 200
```

Map no Javascript: size

A propriedade 'size' funciona como uma espécie de 'length', cuja mesma irá dizer quantos elementos o Map possui. Por exemplo:

```
const frutas = new Map([
  ["maçãs", 200],
  ["bananas", 300],
  ["laranjas", 200]
]);

document.getElementById("demo").innerHTML = frutas.size;
```

3

Map no Javascript: removendo elementos com delete()

O método 'delete()' serve para estar deletando elementos após informar o índice do mesmo. Por exemplo:

```
const frutas = new Map([
  ["maçãs", 200],
  ["bananas", 300],
  ["laranjas", 200]
]);

frutas.delete("bananas");

document.getElementById("demo").innerHTML = frutas.size;
```

2

Map no Javascript: verificando elementos com has()

Com o método 'has()' é possível verificar se determinado elemento existe após ter seu índice informado dentro de parênteses. Caso este elemento exista, o 'has()' retorna 'true', caso não, retorna 'false'. Por exemplo:

```
const frutas = new Map([
  ["maçãs", 200],
  ["bananas", 300],
  ["laranjas", 200]
]);

document.getElementById("demo").innerHTML = frutas.has("laranjas");
```

true

```
const frutas = new Map([
  ["maçãs", 200],
  ["bananas", 300],
]);

document.getElementById("demo").innerHTML = frutas.has("laranjas");
```

false

Diferença de objetos e mapas no Javascript:

	Objeto	Mapa
Iterável	Não é diretamente iterável	Diretamente iterável
Tamanho	Não tem uma propriedade size	Ter uma propriedade size
Tipos de chave	As chaves devem ser Strings (ou Symbols)	As chaves podem ser qualquer tipo de dados
Ordem das chaves	As chaves não estão bem ordenadas	As chaves são ordenadas por inserção
Padrões	Ter chaves padrão	Não tem chaves padrão

Map no Javascript: exibindo todas as chaves com forEach()

Será mostrado a imagem do resultado final primeiro para depois ser explicado em partes:

```
const frutas = new Map([
  ["maçãs", 200],
  ["bananas", 300],
]);

let texto = '';

frutas.forEach(function(valor, indice) {
  texto += indice + ' = ' + valor + '<br>';
});

document.getElementById("demo").innerHTML = texto;
```

maçãs = 200
bananas = 300

Map no Javascript: explicando código do forEach

```
const frutas = new Map([  
    ["maçãs", 200],  
    ["bananas", 300],  
]);
```

Primeiro, é declarado o próprio 'Map' com seus índices e valores, lembrando que todos os elementos do 'Map' vão ficar dentro de um parêntese e colchetes.

Após isso, cada elemento deverá ser coberto de colchetes também.

Map no Javascript: explicando o código do forEach()

```
let texto = '';  
  
frutas.forEach(function(valor, indice) {  
    texto += indice + ' = ' + valor + '<br>';  
});
```

Após declarar o ‘Map’, é declarado uma variável que irá receber os elementos dele.

Logo em seguida, é usado o método ‘forEach()’ com uma função dentro de seus parênteses. Nesta função é informado 2 parâmetros, o valor e o índice (**OBS:** o índice deve estar por último nos parâmetros informados).

Dentro da função, é dito que a variável ‘texto’ irá receber “de forma acumulada (+=)”, o índice, e depois o valor, por um ‘ = ’ concatenado.

Map no Javascript: explicando o código do forEach()

```
document.getElementById("demo").innerHTML = texto;
```

Agora é só exibir a variável ‘texto’, pois ela possui todos os valores com seus respectivos índices do mapa.

Typeof no Javascript:

Existem 5 tipos de dados que podem conter diferentes valores:

- ▶ String
- ▶ Number
- ▶ Boolean
- ▶ Object
- ▶ Function

Typeof no Javascript:

Existem 6 tipos de objetos:

- ▶ Object
- ▶ Date
- ▶ Array
- ▶ String
- ▶ Number
- ▶ Boolean

E existem 2 tipos de dados que não podem conter valores:

- ▶ null
- ▶ undefined

Typeof no Javascript: operador

Para identificá-los, você pode usar o operador 'typeof'. Por exemplo:

typeof "John"	Retorna "string"
typeof 3.14	Retorna "número"
typeof NaN	Retorna "número"
typeof false	Retorna "booleano"
typeof [1,2,3,4]	Retorna "objeto"
typeof {name:'John', age:34}	Retorna "objeto"
typeof new Date()	Retorna "objeto"
typeof function () {}	Retorna "função"
typeof myCar	Retorna "indefinido" *
typeof null	Retorna "objeto"

No caso, apareceu que a variável 'myCar' é indefinido porque ela não recebeu um valor.

Typeof no Javascript: dados primitivos

Um dado primitivo é um tipo de dado que possui apenas seu valor sem qualquer método adicional ou concatenação. Os tipos de dados que podem se tornarem dados primitivos são:

- ▶ String
- ▶ Number
- ▶ Boolean
- ▶ Undefined

Por exemplo:

<code>typeof "John"</code>	Retorna "string"
<code>typeof 3.14</code>	Retorna "número"
<code>typeof true</code>	Retorna "booleano"
<code>typeof false</code>	Retorna "booleano"
<code>typeof x</code>	Retorna "indefinido" (se x não tiver valor)

Typeof no Javascript: dados complexos

O operador ‘typeof’ pode retornar 2 dados complexos, sendo eles:

- ▶ Object
- ▶ Function

O operador retorna ‘object’ para objetos, arrays e null.

Mas o operador não retorna ‘object’ para funções.

Por exemplo:

```
typeof {name:'John', age:34} Retorna "objeto"
typeof [1,2,3,4]           Retorna "objeto" (não "matriz", veja a nota abaixo)
typeof null                Retorna "objeto"
typeof function myFunc(){}

```

Conversão de dados no Javascript:

As variáveis podem ser convertidas em uma nova variável com outro tipo de dado, existem 2 formas de fazer isso:

- ▶ Pelo uso de uma função.
- ▶ Automaticamente pelo próprio javascript.

Convertendo string em número.

O método global 'Number()' serve para converter boa parte dos tipos de string em números. Entre alguns exemplos podemos citar:

- ▶ números entre aspas.
- ▶ espaço vazio (' '), que será convertido em 0.
- ▶ Caso você tente converter uma cadeia de caracteres (como um nome ou coisa assim) em um número, será retornado o valor NaN.

Serão convertidos

```
Number("3.14")  
Number(Math.PI)  
Number(" ")  
Number("")
```

convertidos

```
Number("99 88")  
Number("John")
```

Conversão de dados no Javascript: Métodos globais

Método	Descrição
Number()	Retorna um número, convertido de seu argumento
parseFloat()	Analisa uma cadeia de caracteres e retorna um número de ponto flutuante
parseInt()	Analisa uma cadeia de caracteres e retorna um inteiro

Number faz com que o dado passe para o tipo ‘number’, independente se é um float ou int.

parseFloat faz com que o dado vire um número com casas decimais.

parseInt faz com que o dado se torne um número inteiro.

Conversão de dados no Javascript: operador unário (+)

Caso sua aplicação seja muito grande e você queira fazer as atualizações com a menor quantidade de código para economizar espaço. Uma das medidas seria usar o operador de adição, pois ele também serve para converter uma string em number. Por exemplo:

```
let y = "5";      // y is a string
let x = + y;      // x is a number
```

É criado uma variável do tipo string que contém o número 5, na linha debaixo esta variável é armazenada em outra junto com o operador de adição, e assim, se tornando do tipo 'number'.

OBS: caso tente fazer o mesmo com uma cadeia de caracteres, o valor retornado será NaN.

Conversão de dados no Javascript: number para string

A forma mais comum para utilizar a conversão é através do método global 'String()', onde o valor a ser convertido será passado entre os parênteses. Por exemplo:

```
String(x)      retorna uma cadeia de caracteres de uma variável numérica x  
String(123)    retorna uma cadeia de caracteres de um número literal 123  
String(100 + 23) retorna uma cadeia de caracteres de um número de uma expressão
```

Outra forma é usando o método (não global) 'toString()'. Por exemplo:

```
x.toString()  
(123).toString()  
(100 + 23).toString()
```

O resultado será o mesmo que a imagem de cima.

Conversão de dados no Javascript: boolean para number

Para fazer um tipo de dado ‘boolean’ se tornar um dado ‘number’, basta usar o método global ‘Number()’.

Assim, caso o valor do tipo boolean seja ‘false’, irá retornar 0. Mas caso seja ‘true’, irá retornar 1.

```
Number(false)    // returns 0  
Number(true)     // returns 1
```

Conversão de dados no Javascript: boolean para string

Para realizar uma conversão do tipo de dado 'boolean' para um 'string', basta usar o método global 'String()'. Assim, os valores 'true' e 'false' se tornarão respectivas cadeias de caracteres.

```
String(false)      // returns "false"  
String(true)      // returns "true"
```

O mesmo se aplica ao método 'toString()':

```
false.toString()  // returns "false"  
true.toString()   // returns "true"
```

Expressões regulares no Javascript:

Uma expressão regular é uma sequência de caracteres que forma uma **pesquisa padrão**.

Para facilitar, imagine que você use o método 'search()' para pesquisar por uma palavra, mas não sabe se ela está em letras minúsculas ou qual letra está em maiúsculo. Neste caso, é usado a expressão regular ' /i ', pois ela desabilita o case-sensitive do javascript. Por exemplo:

```
let frase = 'Esta é uma frase RandOmIca';
let pesquisa = frase.search(/randomica/i);
document.write( pesquisa );
```

Expressões regulares no Javascript: algumas expressões

i

Executar correspondência que não diferencia maiúsculas de minúsculas

g

Executar uma correspondência global (localizar todas as correspondências em vez de parar após a primeira partida)

m

Executar correspondência de várias linhas

Expressão regular no Javascript: /g

A expressão regular ‘ /g ’. Serve para fazer uma busca global, ele exibe todas as palavras correspondentes. Além disso, caso use o ‘replace()’, é capaz de substituir todas as palavras automaticamente. Por exemplo:

```
let frase = 'Se o bug buga tudo buga tudo ou somente buga ou bugou mesmo';
let trocar = frase.replace(/bug/g, 'confunde');
document.write( trocar );
```

Se o confunde confundea tudo confundea tudo ou somente confundea ou confundeo mesmo

Expressão regular no Javascript: /m

Esta expressão regular só é aplicada em palavras que foram escritas mais de uma vez. Por exemplo:

```
let frase = 'Expressões regulares';
let repetida = frase.match(/e/m);
document.write( 'Uma letra exibida mais de uma vez é: ' + repetida);
```

Uma letra exibida mais de uma vez é: e

```
let frase = 'Expressões regulares';
let repetida = frase.match(/o/m);
document.write( 'Uma letra exibida mais de uma vez é: ' + repetida);
```

Uma letra exibida mais de uma vez é: null

Expressão regular no Javascript: colchetes

Os colchetes são usados para ajudar em uma pesquisa que tem um intervalo.

Expressão	Descrição
[abc]	Localizar qualquer um dos caracteres entre colchetes
[0-9]	Localizar qualquer um dos dígitos entre colchetes
(x y)	Encontre qualquer uma das alternativas separadas com

Expressão regular no Javascript: [0-0]

O colchete com um traço entre números cita um intervalo entre números. Pode usar isso para buscar uma determinada quantidade de caracteres. Por exemplo:

```
let texto = "123456789";
let resultado = texto.match(/[1-4]/g);
document.getElementById("demo").innerHTML = resultado;
```

1,2,3,4

Expressões regulares no Javascript: (x | y)

Os parênteses com uma barra no meio (|) serve para pesquisar por duas palavras, para caso não encontre uma, exiba a outra.

```
let texto = "ve, verde, vermelho, verde, verd, de, azul, amarelo";
let resultado = texto.match(/(vermelho|verde)/g);
document.getElementById("demo").innerHTML = resultado;
```

verde,vermelho,verde

Expressões regulares no Javascript: MetaCaracteres

MetaCaracteres são caracteres com um significado especial. Você costuma usá-los dentro de uma expressão regular. Podemos citar alguns deles. sendo:

Metacaractere	Descrição
\d	Localizar um dígito
\s	Localizar um caractere de espaço em branco
\b	Localizar uma correspondência no início de uma palavra como esta: \bWORD ou no final de uma palavra assim: WORD\b
\uxxxx	Localize o caractere Unicode especificado pelo número hexadecimal xxxx

Expressões regulares no Javascript: quantificadores

Quantificadores são usados para exibir todos os caracteres especificando apenas um dentro de uma variável. Podemos citar 3 deles, sendo:

Quantificar	Descrição
n^+	Corresponde a qualquer cadeia de caracteres que contenha pelo menos um n
n^*	Corresponde a qualquer cadeia de caracteres que contenha zero ou mais ocorrências de n
$n^?$	Corresponde a qualquer cadeia de caracteres que contenha zero ou uma ocorrência de n

Try e Catch no Javascript:

Caso você tenha tentando escrever algum código no javascript e por descuido cometeu algum erro, percebeu que o código todo que estava abaixo deste erro parou de funcionar também?

Tal ocorrência pode vir a se tornar um problema enorme. Por exemplo: você decide colocar um site no ar que armazena dados de vários usuários em tempo real. Mas caso uma pessoa digite algo que dispare um erro no Javascript, o código todo abaixo do erro vai parar de funcionar.

Para contornar esta situação, foi criado os comandos 'try' e 'catch' para analisarem códigos que possuem alguma possibilidade de dispararem um erro, e informa ao javascript como deve tratá-los.

Try e catch no Javascript: sintaxe

A sintaxe correspondente destes comandos são:

```
try{  
    // código que pode dar errado  
} catch (parâmetro que pode ser qualquer coisa) {  
    // o que fazer com o erro que foi encontrado  
}
```

Try e catch no Javascript: prática

Para testar esses dois comandos, foi resolvido escrever a palavra 'alert(para exibir um pop-up com a mensagem desejada)' de forma errada. O resultado foi esse:

```
try {
    adddlert("Bambu Africano");
}
catch(err) {
    document.getElementById("demo").innerHTML = err.message;
}
```

adddlert is not defined

O parâmetro do 'catch' está definido como 'err' mas ele pode ser o que você quiser, apenas é o recomendado deixá-lo assim.

O método 'message' no 'err' fará com que seja exibido a mensagem de erro que apareceria no console da página.

Try e catch no Javascript: throw

Caso você queira disparar mensagens **próprias** de erro, pode usar o comando 'throw', dentro do 'try' em uma verificação. Por exemplo:

Caso você tente converter uma string que o usuário enviar em um número e der NaN:

```
<button id="btn" onclick="conversao()">testando</button>
<p id="p01"></p>

<script>

    function conversao() {
        let valor = prompt('Informe um valor');

        let numero = parseInt(valor);
        document.getElementById('p01').innerHTML = "O número digitado é: " + numero;

        if (!isNaN(numero)) {
            throw Error('Digite apenas números');
        }
    }

    try {
        conversao();
    } catch(err) {
        alert('Ocorreu um erro: ' + err.message);
    }
</script>
```

Entendendo o código do try e catch

```
function conversao() {  
    let valor = prompt('Informe um valor:');  
  
    let numero = parseInt(valor);  
    document.getElementById('p01').innerHTML = "O número digitado é: " +  
    numero;
```

Primeiro, criamos uma função dentro do javascript e atribuímos o valor que o usuário mandar na variável 'valor'.

Na linha de baixo, criamos uma variável que irá converter automaticamente o valor armazenado para um número, pois **TODO** dado que o usuário enviar será uma string.

Na linha abaixo nós iremos escrever no parágrafo que tiver o id 'p01' a mensagem que está acima.

Entendendo o código do try e catch

```
if (isNaN(numero)) {  
    throw Error('Digite apenas números');  
}
```

Ainda dentro da função ‘conversao’, nós verificamos se o ‘**numero**’(aquele que vai converter o valor para number)’ é um NaN, o que significa que o usuário enviou uma cadeia de caracteres.

Caso ele seja um NaN, vai disparar um erro através do comando ‘throw Error()’. No entanto, o código não vai funcionar se fizer apenas isso.

Entendendo o código do try e catch

```
try {
    conversao();
}
catch(err) {
    alert('Ocorreu um erro: ' + err.message);
}
```

Então, fora da função, escrevemos o comando ‘try’ e ‘catch’. Onde ‘try’ irá tentar rodar essa mesma função na hora em que o usuário ativa-la, e capturar o erro.

Então na linha abaixo, ‘catch’ irá exibir esse erro em forma de um pop-up passo pelo alert, exibindo exatamente a mensagem que informamos pelo throw.

Motivos para usar o throw:

Um dos principais motivos é quando você for criar uma função que seja capaz de otimizar alguma tarefa para seus amigos, ou até mesmo para o público em geral.

Deve-se estar ciente de que nem todos pensam como você, podendo acabar com dificuldades em entender esta função.

Então algumas mensagens de erro personalizadas que expliquem diretamente o que está errado de forma simples irá ajudar imensamente a todos.

AVISO DE BOA PRÁTICA: declaração de variáveis

Em todo escopo, sendo ele uma função, ou mesmo no início do código. Sempre declare variáveis no começo do escopo (local e global) pois isso evita o javascript de gerar algum bug por ‘variável não identificada’ caso você a utilize antes de declará-la por engano.

Declarando o modo estrito no Javascript:

O modo estrito é feito justamente para evitar práticas ruins. O exemplo mais comum de sua utilidade é usando uma variável que não foi declarada com 'var', 'let' ou 'const', todavia, ele interrompe o navegador de continuar rodando seu código após encontrar este erro.

Para usar o modo estrito basta escrever no começo do código: "use strict"(uso estrito).

```
<script>
  "use strict";
  x = 3.14; // Este código não será executado, pois a variável x não foi
            declarada.
</script>
```

Declarando o modo estrito no Javascript: escopos

O modo estrito depende do escopo em que ele se encontra, dessa forma, ele será aplicado no código inteiro caso seja escrito no topo do script. Mas ele só irá funcionar em uma única função caso seja declarado apenas no começo dela. Por exemplo:

```
<script>
x = 3.14;      // Não vai causar um erro.
myFunction();

function myFunction() {
    "use strict";
    y = 3.14;  // Vai causar um erro (y não está definido).
}
</script>
```

Declarando o modo estrito no Javascript: motivos para usá-lo

Ele foi feito para tornar seu código mais compatível com as versões anteriores, que tinham uma sintaxe rigorosa e impediam o código de rodar no menor dos erros.

Assim, o que o javascript normalmente permite usar sem problemas, como uma variável não declarada (não importando se é um tipo de dado primitivo ou composto) é definido como um erro pelo ‘modo estrito’, para prevenção de futuros bugs.

OBS: O modo estrito só é reconhecido no javascript caso seja declarado no começo do código ou de uma função, nunca no meio ou no final.

Coisas não permitidas no modo estrito

Não é possível usar dados primitivos ou compostos sem antes criá-los.

Não é possível deletar uma variável depois de usá-la (tanto primitiva quanto composta).

```
"use strict";
let x = 3.14;
delete x;      // Vai dar um erro.
```

Não é permitido usar o mesmo nome de um parâmetro mais de uma vez dentro dos parênteses:

```
"use strict";
function x(p1, p1) {};  // Vai dar um erro, pois o parâmetro 'p1' foi
declarado 2 vezes
```

Arrow function no Javascript:

Uma ‘arrow function’, também conhecida como “função anônima”. É extremamente útil quando seu objetivo for economizar código e poupar a memória RAM para o site ser exibido ao usuário.

Arrow function no Javascript: características

Agora vamos falar de suas principais características e o porquê de usá-lo:

- ▶ Não é preciso usar a palavra-chave 'function', pois o javascript já entende que se trata de uma arrow function.
- ▶ Por ser uma função anônima, ele não tem um nome de função, logo, ele é armazenado dentro de uma variável: `let variavel = () => {};`
- ▶ O return é desnecessário na arrow function, pois o javascript já entende que você pretende retornar o valor.

```
let variavel = (num1, num2) => {  
    num1 + num2;  
};
```

OBS: Por se tratar de uma função com o objetivo de simplificar código, ele só é recomendado em casos que seja passado apenas uma instrução, mas pode usá-lo com mais, obviamente.

Arrow function no Javascript: abreviações

Caso você escreva apenas **uma** instrução na arrow function, o uso de chaves ({}) é desnecessário:

```
let variavel = (num1, num2) => num1 + num2;
```

Caso você informe apenas **um** parâmetro, o uso de parênteses também é desnecessário:

```
let variavel = idade => idade >= 18;
```

No entanto, não é tão recomendado tirar os parênteses, uma vez que pode acabar atrasando ou confundido a manutenção do código.

Importando e exportando pedaços de código no Javascript

O javascript consegue estar importando pedaços de códigos de outros arquivos '.js' e utilizar no seu código atual sem quaisquer problemas.

Estes arquivos importados são funções, pois elas não irão atrapalhar o restante do código e podem ser utilizadas quando quiserem.

Exportando funções no Javascript

Para exportá-las, você precisa escrever a palavra 'export' antes de declará-las. Por exemplo:

```
export function somar(n1, n2) {  
    return n1 + n2;  
}
```

OBS: Você pode exportar quantas funções quiser.

Importando funções no Javascript

Agora para importá-las no seu código atual. Utilize a seguinte sintaxe:

```
import {nomeFuncao} from './nomeArquivo.js'
```

Por exemplo:

```
import { multiplicar } from './script2.js';
```

O import ignora espaços e quebras de linhas dentro das chaves{}, então também pode escrevê-los da seguinte forma:

```
import {  
    multiplicar  
} from './script2.js';
```

Definindo o tipo de script

Como você importou e exportou pedaços de código, coisas que não são habituais do Javascript fazer (mas sim o React). Você terá de definir o tipo do script, usando o atributo e valor `type="module"` para não gerar qualquer erro. O código deverá ser escrito de forma semelhante a esta:

```
<script type="module" src="script.js"></script>
```

OBS: a ordem dos atributos não é relevante.

Boas práticas de arquivos JS: variáveis

Usamos como boa prática no Javascript fazer a separação entre o nome das variáveis através do camelCase. Onde cada palavra será separada por sua letra maiúscula, da seguinte forma:

```
firstName = "John";  
lastName = "Doe";
```

Esta regra se aplica tanto para nome de variáveis como também de funções.

Boas práticas de arquivos JS: operadores

É uma boa prática sempre deixar um espaço entre cada operador (= / * + -). Para facilitar a visualização de outras pessoas. Por exemplo:

```
let x = y + z;  
const myArray = ["Volvo", "Saab", "Fiat"];
```

Boas práticas de arquivos JS: redução de código

Caso prefira não usar a tecla 'tab', já que ela gasta muito espaço da tela, pode apertar a tecla de espaço **duas vezes** para substituí-lo. É sempre bom utilizá-lo em blocos de código como funções, etc:

```
function toCelsius(fahrenheit) {  
    return (5 / 9) * (fahrenheit - 32);  
}
```

Boas práticas de arquivos JS: regras gerais

É bem visto para todos, utilizar o ponto e vírgula (;) sempre que terminar uma instrução, não apenas quando escrever duas instruções na mesma linha.

```
const cars = ["Volvo", "Saab", "Fiat"];
```

Existem também algumas regras gerais para instruções complexas (funções, loops, desvio condicional), sendo elas:

- ▶ Coloque o colchete de abertura na linha em que a instrução é declarada.
- ▶ Use um espaço antes do colchete de abertura.
- ▶ Coloque o colchete de fechamento em uma nova linha.
- ▶ Não termine uma instrução complexa com ponto e vírgula.

```
function toCelsius(fahrenheit) {  
    return (5 / 9) * (fahrenheit - 32);  
}
```

Boas práticas de arquivos JS: regras para objetos

Há também algumas regras específicas para objeto. Sendo elas:

- ▶ Coloque o colchete de abertura na mesma linha que o nome do objeto.
- ▶ Utilize um espaço entre o nome da propriedade e seu valor.
- ▶ Use aspas em torno de valores de cadeia de caracteres, e não de numéricos.
- ▶ Não utilize uma vírgula na última propriedade-valor.
- ▶ Coloque o colchete de fechamento em uma nova linha.
- ▶ Sempre termine uma declaração de objeto com ponto e vírgula.

```
const person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50,  
    eyeColor: "blue"  
};
```

Boas práticas de arquivos JS: regras para objetos

Caso o objeto possua uma quantidade pequena de propriedades-valor, pode escrever toda a declaração em uma única linha. Por exemplo:

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Boas práticas de arquivos JS: Comprimento da linha < 80

Para facilitar a visualização do código, é recomendado que cada linha tenha menos de 80 caracteres. Para isso, você pode quebrar a linha após um operador ou uma vírgula.

```
document.getElementById("demo").innerHTML =  
    "Hello Dolly.";
```

Boas práticas de arquivos JS: nomenclatura

Sempre use o mesmo tipo de nomenclatura para todo seu código, para que ele seja padronizado e evite futuros erros. Por exemplo:

- ▶ Nome de variáveis e funções escrito com **camelCase** (Ex: nomeCompleto).
- ▶ Variáveis globais escritas em maiúscula (é bastante comum, mas ainda existem algumas pessoas que não a utilizem, a propósito, variável global é uma que não pertence a um escopo de bloco).
- ▶ Constantes (como PI) também são escritas em maiúsculas.

No entanto, alguns programadores preferem usar o método **pascal_case** que troca os espaços por sublinhados. (Ex: nome_completo). Esta alternativa é mais usada em linguagens como o C.

Boas práticas de arquivos JS: carregando o script

Sempre que for usar um código de script externo, procure sempre usar a forma mais simples, como esta:

```
<script src="myscript.js"></script>
```

OBS: O atributo 'type' só é necessário caso seu arquivo JS importe algo de algum outro arquivo.

Nota: Lembre-se de tomar cuidado com o case-sensitive do javascript.

Boas práticas de arquivos JS: evite usar variáveis globais

Evite usar variáveis que sejam declaradas pela palavra-chave ‘var’. Pois elas podem ser recriadas ou terem seus valores alterados, isto se aplica tanto em variáveis como também funções e objetos.

Boas práticas de arquivos JS: declarando variáveis no topo

Declarar variáveis no topo de cada escopo é uma excelente prática, pois ajuda na legibilidade, além de dar ao usuário apenas 1 lugar para procurar e encontrar todas variáveis daquele escopo. Por exemplo:

```
Declarar no início
let firstName, lastName, price, discount, fullPrice;

Utilizar mais tarde
firstName = "John";
lastName = "Doe";

price = 19.90;
discount = 0.10;

fullPrice = price - discount;
```

Boas práticas de arquivos JS: declarar objetos com const

Declarar um objeto com ‘const’ impedirá que qualquer um de seus valores sofra uma mudança indesejada.

```
let cars = ["Saab", "Volvo", "BMW"];
cars = 3;    Altera a matriz para o número
```

```
const cars = ["Saab", "Volvo", "BMW"];
cars = 3;    Não é possível
```

Como pode ver, a mesma boa prática também é recomendada em arrays.

Boas práticas de arquivos JS: conversões automáticas

Deve-se tomar muito cuidado com as conversões automáticas, que ocorrem simplesmente por você alterar o valor de uma variável.

```
let x = "Hello";
x = 5;
```

Boas práticas de arquivos JS: terminando switch com default

É recomendável sempre terminar o ‘switch’ usando o ‘default’, pois sabemos que nem sempre o usuário vai digitar uma das opções apresentadas. Neste caso, seu código poderá quebrar em algum momento caso não saiba tratar deste erro.

```
default:  
    day = "Unknown";
```

Melhorando desempenho em arquivos JS: loops

Caso você utilize um loop para percorrer um array e use o método 'nomeArray.length' para defini-lo como um limite, saiba que colocar este método na verificação do loop o deixará mais lento, pois ele terá que fazer o método **de novo** sempre que o loop se repetir. Para arrumar este defeito, basta criar uma variável que irá receber este método. Por exemplo:

```
let l = arr.length;  
for (let i = 0; i < l; i++)
```

Melhorando desempenho em arquivos JS: acessando o DOM

Acessar o DOM é o mesmo que escrever 'document.nomeMetodo' para manipular o body. O processo de acesso ao body do html é um pouco lento, então usá-lo várias vezes no seu código naturalmente irá o deixar devagar.

Então, caso vá manipular o DOM mais de 2 ou 3 vezes, basta colocá-lo apenas 1 vez dentro de uma variável, e assim, substituindo DOM com ela. Por exemplo:

```
const obj = document.getElementById("demo");
obj.innerHTML = "Hello";
```

Melhorando desempenho em arquivos JS: tamanho do DOM

Caso você acabe desenvolvendo a prática de fazer as operações no DOM para exibir elas assim que estiverem concluídas, saiba que não é uma prática recomendada, pois ela pode acabar desacelerando o seu site. Já que o navegador terá de interromper a instrução de manipular o DOM para poder resolver a expressão e conseguir mostrá-la na manipulação.

Para resolver isso, basta criar uma variável que receberá a expressão, e então colocá-la no código que irá alterar o <body>.

```
let fullName = firstName + " " + lastName;  
document.getElementById("demo").innerHTML = fullName;
```

Projetos para praticar o Javascript (obrigatório):

1. Country Guide App | Javascript API Project

Link: <https://www.youtube.com/watch?v=QDCmQHO8F8Q>

2. To Do List With Javascript | Step by Step Javascript Project

Link: <https://www.youtube.com/watch?v=cOUNOi297Mw>

3. QR Code Generator in HTML CSS & JavaScript | QR Code Generator in JavaScript

Link: <https://www.youtube.com/watch?v=pv5K28zVepE>

4. Random Password Generator in HTML CSS & JavaScript

Link: <https://www.youtube.com/watch?v=fD8gm-DhjXk>

5. Pokemon Card Generator Javascript | Step By Step Javascript Project

Link: https://www.youtube.com/watch?v=_JUSpgchD1I

6. 3D Cube Image Slider | HTML, CSS And Javascript

Link: https://www.youtube.com/watch?v=B_qfQuBmfiw

7. Build A Memory Card Game in HTML CSS & JavaScript

Link: <https://www.youtube.com/watch?v=DABkhfsBAWw>

8. Box Shadow Generator | HTML, CSS & Javascript | Step By Step JS Project

Link: <https://www.youtube.com/watch?v=9WZ4ajDNmrU>

Projetos para praticar o Javascript (obrigatório):

1. Create A Snake Game in HTML CSS & JavaScript | JavaScript Game Tutorial

Link: <https://www.youtube.com/watch?v=K8Rh5x3c9Pw>

2. Drawing App With Javascript | Step by Step Javascript Project

Link: <https://www.youtube.com/watch?v=gulH9uC5Zml>

3. Create A Random Color Palette Generator in HTML CSS & JavaScript

Link: <https://www.youtube.com/watch?v=H-LvaBNLDSQ>

4. Typing Speed Test Game in HTML CSS & JavaScript

Link: <https://www.youtube.com/watch?v=Hg80AjDNnJk>

5. Build A Weather App in HTML CSS & JavaScript | Weather App in JavaScript

Link: <https://www.youtube.com/watch?v=Hg80AjDNnJk>

6. Get User Location Using Javascript | HTML, CSS & Javascript

Link: <https://www.youtube.com/watch?v=wJ6f2Gi7sC8>

7. Memes App | HTML, CSS & JS | Javascript Project With Source Code

Link: <https://www.youtube.com/watch?v=H0Zb22HpRuw>

Projetos para praticar o Javascript (opcional):

1. Build A Notes App in HTML CSS & JavaScript | Notes App in JavaScript

Link: <https://www.youtube.com/watch?v=AklUfUWpyZs>

2. Create A Dynamic Calendar in HTML CSS & JavaScript | Calendar in JavaScript

Link: <https://www.youtube.com/watch?v=Z1BGAivZRIE>

3. Create A Custom Select Menu with Search Box in HTML CSS & JavaScript

Link: <https://www.youtube.com/watch?v=z0avfnlBRto>

4. Build A Currency Converter App in HTML CSS & JavaScript

Link: <https://www.youtube.com/watch?v=UY7F37Khyl8>

5. Custom Emoji Range Slider using HTML CSS & JavaScript | Mood Slider in JavaScript

Link: <https://www.youtube.com/watch?v=RfPXF1W01fU>

Projetos para praticar o Javascript (opcional):

1. Easy Digital Clock | HTML, CSS & Javascript | Javascript Project

Link: <https://www.youtube.com/watch?v=JYDxTt3lh0>

2. Blob Maker | Step By Step Javascript Project With Source Code

Link: <https://www.youtube.com/watch?v=jiqYW8GHmas>

3. Calculator Using HTML, CSS And Javascript | With Source Code

Link: https://www.youtube.com/watch?v=hma0N8Vu_Uw

4. Random Number Between A Given Range | HTML, CSS & Javascript

Link: <https://www.youtube.com/watch?v=4qG4ADVcg1o>

5. Random User Card Generator | Javascript Project With Source Code

Link: <https://www.youtube.com/watch?v=pCxVSWJ6L7g>

Projetos para praticar o Javascript (opcional):

1. Color The Parrot | Javascript Project With Source Code

Link: <https://www.youtube.com/watch?v=gpkD2Hsw-og>

2. Interactive Panda Form | HTML, CSS & Javascript Project

Link: <https://www.youtube.com/watch?v=jQmuuhMy7g>

3. Dog Makeover Game | HTML, CSS & Javascript Project

Link: <https://www.youtube.com/watch?v=R5PrHGgrv5g>