



BOSCH
Invented for life

Aula 1 - Introdução

Docupedia Export

Author: Siqueira Joao (CtP/ETS)
Date: 22-Dec-2022 12:23

Table of Contents

1	O que é o JavaScript?	4
2	Como funciona	5
3	Vantagens	6
4	Desvantagens	6
5	Console.log	8
6	Comentários	9
7	Mesclando com HTML	10
8	Node vs navegador	11
9	Variáveis e Constantes	12
10	Var	13
11	Let	13
12	Operadores aritméticos	14
13	Operadores de comparação	14
14	Objeto Window	15
15	Alert, Confirm e Prompt	16
16	Mudar tipo da variável	18
17	Funções com números	19
18	Operações com o objeto Math	21
19	Valores Falsy e Truthy em JavaScript	22
20	Operação ternária	23
21	Arrays	24
22	Exercícios	31
23	Correção	32
24	DOM	34



1 O que é o JavaScript?

É uma linguagem de programação interpretada estruturada, de script em alto nível com tipagem dinâmica fraca e multiparadigma.

Segundo a Mozilla Foundation, atual nome da antiga Netscape Communications Corporations, empresa responsável pela criação do JS, *"JavaScript é uma linguagem de programação, leve, interpretada, orientada a objetos, baseada em protótipos e em first-class functions (funções de primeira classe), mais conhecida como a linguagem de script da Internet."*

O JavaScript é uma das mais importantes tecnologias voltadas para o front-end e, unindo-se ao trio HTML, CSS e PHP, formam um grupo de linguagens que abrangem praticamente todas as exigências do desenvolvimento de uma página completa, dinâmica e com boa performance.

Alguns exemplos de sites que utilizam JS em seu front e back-end hoje em dia são **Ebay**, **LinkedIn** e **Yahoo**.

Mas o JS não se restringe mais apenas às páginas e aos navegadores, como foi durante vários anos: com o advento de diversos frameworks, APIs, melhorias e criação de centenas de funções, hoje já é possível utilizar JavaScript em aplicativos mobile, softwares para desktop e até mesmo em back-end.

2 Como funciona

É uma linguagem de programação client-side, ou seja, é executada do lado do usuário, mais especificamente pelo navegador utilizado por este usuário.

Em outras palavras, isso significa que todas as suas ações são processadas na máquina de quem as utiliza, sem a necessidade de enviá-las a nenhum outro ambiente.

Como nada é enviado a nenhum servidor externo para processamento, as respostas são imediatas.

3 Vantagens

- Versatilidade da linguagem;
- Rapidez de leitura e, portanto, rapidez de execução;
- Sintaxe acessível;
- Não precisa ser compilada — ou seja, os navegadores são capazes de interpretá-la por conta própria;
- Ótima linguagem para iniciantes em programação;
- Compatível com uma grande variedade de navegadores e plataformas;
- Código leve;
- Curva de aprendizagem rápida;
- Grande comunidade ao redor do mundo.

4 Desvantagens

Como tudo tem dois lados, o JS também conta com algumas desvantagens.

Entre elas, estão:

- Poucos recursos voltados à segurança;
- Pode conter brechas para a execução de ações maliciosas;
- O Node.js está propenso a mais vazamento de memória em processos de execução longa.

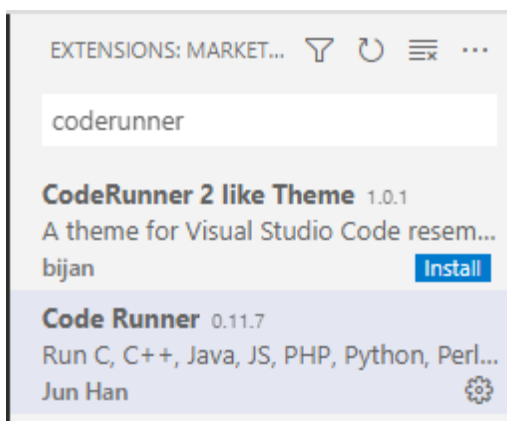
Fundamentos do JS

Ferramentas:



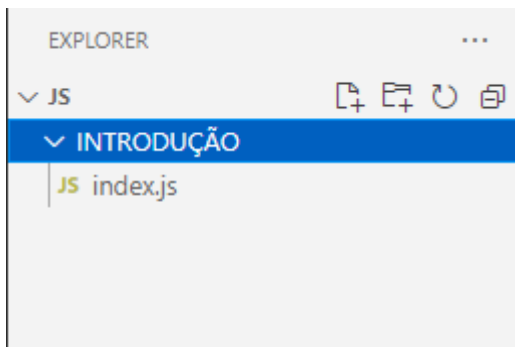
Para iniciarmos abra o VS Code.

Verifique se a extensão CodeRunner está instalada:



Caso não esteja instale para que seja possível executar o código diretamente com node.js.


Abra ou crie uma pasta para o curso, abra esta no VS Code, dentro dela uma crie um arquivo index.js



5 Console.log

É uma função para saída de dados no console, muito utilizada para debugs, não interfere na execução do conteúdo no navegador.

```
1 console.log('Curso JS');  
2 console.log("Curso JS");  
3 console.log(`Curso JS`);  
4 console.log(55,48.9,'Curso JS');
```

Para executar o código clique no ícone  ou Ctrl + Alt + N

```
[Running] node "u:\JS\INTRODUÇÃO\index.js"
```

```
Curso JS
```

```
Curso JS
```

```
Curso JS
```

```
55 48.9 Curso JS
```

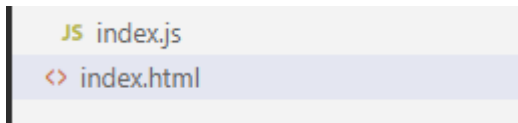
```
[Done] exited with code=0 in 0.494 seconds
```


6 Comentários

```
1 //Duas Barras para comentar uma linha
2
3 /* Barra
4 e asterisco
5 para bloco de comentário*/
```

7 Mesclando com HTML

Crie um arquivo .html.



Podemos criar scripts junto a um código html utilizando a tag `<script></script>`

```
1  <!DOCTYPE html>
2  <html lang="pt">
3  <head>
4      <meta charset="UTF-8">
5      <title>Document</title>
6  </head>
7  <body>
8
9      <script>
10         console.log("Curso JS")
11     </script>
12 </body>
13 </html>
```

É possível incluir a tag no `<head>` como também no `<body>`, porém tudo que está dentro do `<head>` é executado antes do carregamento da página. Incluir o script pode retardar esse carregamento, por isso, é uma boa prática colocar o script dentro do body ao fim do código html, assim a o script será executado após o carregamento da página otimizando esse processo.

Porém incluir esse script no mesmo arquivo html, pode deixá-lo muito longo e até mesmo confuso, dessa forma uma boa o indicado é criar um arquivo .js e incluir o link entre os dois arquivos, também dentro da tag `<body>` ao fim do código html.

```
1  <!DOCTYPE html>
2  <html lang="pt">
3  <head>
4      <meta charset="UTF-8">
5      <title>Document</title>
6  </head>
7  <body>
8
9      <script src="index.js"></script>
10 </body>
11 </html>
```

8 Node vs navegador

Existem diferença ao utilizarmos o javascript no front-end e no back-end. Por exemplo a função alert:

```
1 alert('node e navegador não são a mesma coisa')
```

Quando executada no navegador gera um alerta que para a execução da página até que o usuário clique em OK.

Essa página diz

node e navegador não são a mesma coisa

OK

Já no node.js essa função não é reconhecida e gera um erro.

```
[Running] node "u:\JS\INTRODUÇÃO\tempCodeRunnerFile.js"
u:\JS\INTRODUÇÃO\tempCodeRunnerFile.js:1
alert('node e navegador não são a mesma coisa')
^
```

```
ReferenceError: alert is not defined
    at Object.<anonymous> (u:\JS\INTRODUÇÃO\tempCodeRunnerFile.js:1:1)
    at Module._compile (node:internal/modules/cjs/loader:1103:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1155:10)
    at Module.load (node:internal/modules/cjs/loader:981:32)
    at Function.Module._load (node:internal/modules/cjs/loader:822:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:77:12)
    at node:internal/main/run_main_module:17:47
```

```
[Done] exited with code=1 in 1.07 seconds
```

Assim podemos concluir que são dois ambiente diferentes com funções diferentes também. Para o front-end utilizamos com intuito de manipular divs e elementos das páginas, já no back-end utilizamos para trabalhar com base de dados e rotas de aplicação.

9 Variáveis e Constantes

O Javascript é uma linguagem de tipagem dinâmica, ou seja, não é necessário declarar o tipo de dados ao criar variáveis ou constantes.

Para definirmos constantes utilizamos "const".

1

```
const pi= 3.14159265359;
```

Já para definirmos variáveis temos "var" e "let", existem algumas diferenças na aplicação das duas.

10 Var

- Com var as variáveis podem ser definidas inúmeras vezes.

1	<code>var nome = 'Geraldo'</code>
2	<code>var nome = 'Antônio'</code>

- Var tem escopo de função.

11 Let

- Com let a variável deve ser definida uma única vez. Caso contrário ocorrerá um erro.

1	<code>let nome = 'Geraldo'</code>
---	-----------------------------------

- Let tem escopo de bloco.

```

1  const verdadeira = true;
2
3  let nome = "Luiz"; //criando
4  var nome2 = "Alberto"; //criando
5
6  if(verdadeira){
7      let nome = "Joana"; //criando
8      var nome2 = 'Catarina'; //redeclarando
9      if(verdadeira){
10         var nome2 = 'Outro nome'; // redeclarando
11         let nome = 'Roberto'; //criando
12     }
13 }
14 console.log(nome,nome2);
15
16 function sobrenome(){
17     var sobrenome = 'Fredericksen';
18 }
19 console.log(sobrenome); // erro, pois a variável foi declarada apenas na
    função.
```

Tipos de dados



12 Operadores aritméticos

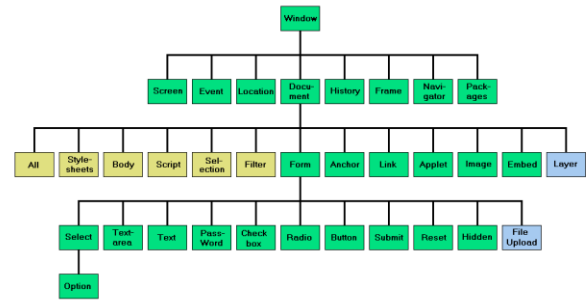
+	Soma valores
-	Subtrai valores
*	Multiplica valores
/	Divide valores
%	Resto da divisão
++	Incremento de 1
--	Decremento de 1

13 Operadores de comparação

==	Igual
!=	Diferente
<>	Diferente
===	Idêntico
!==	Não idêntico
<	Menor que
>	Maior que
<=	Menor ou igual
>=	Maior ou igual

14 Objeto Window

É o principal ponto de entrada para todos os recursos e APIs de Javascript do lado do cliente. Ele representa uma janela ou quadro de navegador Web e pode ser referenciado através do identificador window. Em Javascript do lado do cliente, o objeto Window também é o objeto Global. Isso significa que o objeto Window está no topo do encadeamento de escopo e que suas propriedades e métodos são efetivamente variáveis globais e funções globais.

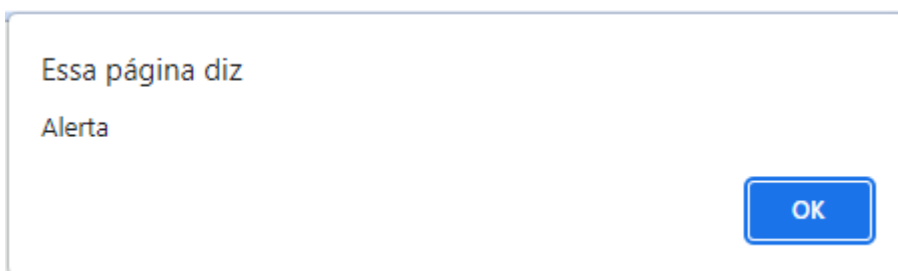


15 Alert, Confirm e Prompt

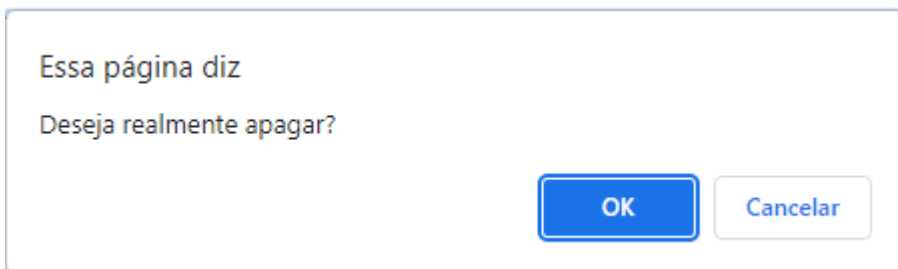
São funções do objeto window executa

Alert como já vimos anteriormente interrompe a execução de qualquer função na página e exibe um alerta com a mensagem programada.

```
> window.alert('Alerta')  
< undefined  
>
```



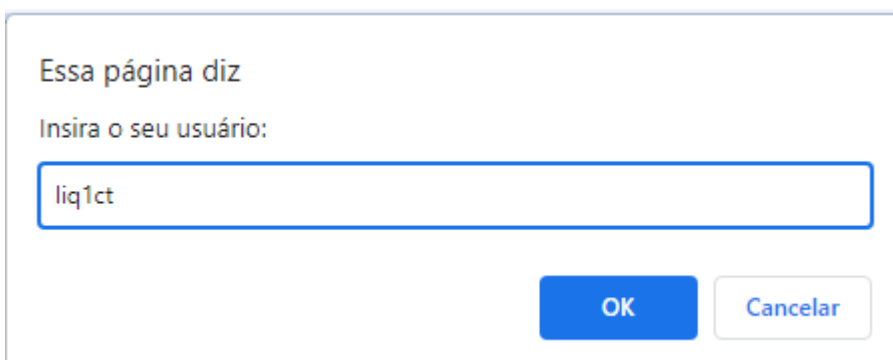
Da mesma forma temos a função confirm que retorna uma resposta (true/false) e permite outros eventos a partir dessa resposta.



```
> window.confirm('Deseja realmente apagar?')  
< true  
>
```

Ainda temos a função prompt que permite uma entrada de texto.

```
> window.prompt('Insira o seu usuário:')
```



E retorna a inserção do usuário.

```
> window.prompt('Insira o seu usuário:')  
< 'liq1ct'  
>
```

16 Mudar tipo da variável

Ao retornar um dado ele vem em forma de string

```
> let num = prompt("Digite um número");  
↵ undefined  
  
> typeof(num)  
↵ 'string'
```

Então em alguns caso é preciso mudar o tipo da variável para number

<pre>> num = parseFloat(num) ↵ 50 > typeof(num) ↵ 'number'</pre>	<pre>> num = Number(num) ↵ 50 > typeof(num) ↵ 'number'</pre>
---	---

17 Funções com números

Algumas funções simples que podemos fazer com números são de transformar em string ou binário, arredondar o número para quantas casas decimais forem necessárias e verificar se é um inteiro ou NaN

```
1 let num1 = 10;
2 let num2 = NaN;
3
4 console.log(num1.toString()); // Mostra o número em string
5 console.log(num1.toString(2)); // Mostra o número em binário
6 console.log(num1.toFixed(2)); // Arruma as casas decimais do número
7 console.log(Number.isInteger(num1)); // Verifica se o número é inteiro
8 console.log(Number.isNaN(num2)); // Verifica se o número é NaN
```

[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"

10

1010

10.00

true

true

[Done] exited with code=0 in 0.555 seconds

Por conta do padrão que o Java Script segue para ter a precisão dos números, pode haver uma leve imprecisão quando trabalhamos com casas decimais

```
1 let num1 = 0.7;
2 let num2 = 0.1;
3
4 console.log(num1 + num2); // Há imprecisão
5
6 let num3 = (num1 + num2).toFixed();
7 console.log(num3);
8 console.log(Number.isInteger(num3)); // Apenas usar o "toFixed" não
   funciona
9
10 num3 = Number(num3)
11 console.log(num3);
12 console.log(Number.isInteger(num3)) // É preciso transformar em Number de
   novo
```

[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"

0.7999999999999999

1

false

1

true

[Done] exited with code=0 in 0.546 seconds

18 Operações com o objeto Math

Podemos fazer operações matemáticas também com o objeto Math

```
1 let num = 9.548
2
3 console.log(Math.floor(num)) // Arredonda o número para baixo
4 console.log(Math.ceil(num)) // Arredonda o número para cima
5 console.log(Math.round(num)) // Arredonda o número
6 console.log(Math.random()) // Gera um número aleatório entre 0 e 1
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
9
10
10
0.439589305481376
```

```
[Done] exited with code=0 in 0.579 seconds
```

Em Java Script é possível dividir por 0

```
1 let num = (100 / 0) // Essa conta é válida e tem como resultado "Infinity"
2
3 console.log(num)
4 console.log(typeof(num))
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
Infinity
number
```

```
[Done] exited with code=0 in 0.524 seconds
```

19 Valores Falsy e Truthy em JavaScript

Tudo em JavaScript pode ser avaliado em falso ou verdadeiro, dessa forma temos os valores **FALSY**, tirando esses valores todos os outros são considerados verdadeiros



```

1 console.log(Boolean(false));
2 console.log(Boolean(NaN));
3 console.log(Boolean(undefined));
4 console.log(Boolean(null));
5 console.log(Boolean(""));
6 console.log(Boolean(''));
7 console.log(Boolean(0));
8
9 // Todos retornam False como valor

```

[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
false
false
false
false
false
false
false
false
false
[Done] exited with code=0 in 0.536 seconds

Em uma comparação é possível fazer retornar o valor falso, ou caso não haja retornar o ultimo valor verificado

```

1 console.log("Nome" && 25 && 2.88 && "Esse vai ser mostrado");
2 console.log("Nome" && 25 && 2.88 && undefined && "Esse não vai ser mostrado");

```

[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
Esse vai ser mostrado
undefined
[Done] exited with code=0 in 0.541 seconds

20 Operação ternária

A Operação ternária ajuda a criar condições para variáveis de forma mais rápida

```
1  const pontuacaoUsuario = 1000;  
2  const tipoUsuario = pontuacaoUsuario >= 1000 ? 'Usuário VIP' : 'Usuário  
normal'; // (condição) ? (valor verdadeiro) : (valor falso)  
3  console.log(tipoUsuario);  
4  
5  var corUsuario = null;  
6  var corPadrao = corUsuario || 'Azul';  
7  console.log(corPadrao);  
8  
9  var corUsuario = 'Roxo';  
10 var corPadrao = corUsuario || 'Azul';  
11 console.log(corPadrao);
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"  
Usuário VIP  
Azul  
Roxo  
  
[Done] exited with code=0 in 0.566 seconds
```

21 Arrays

Para adicionar e remover itens dos arrays

```
1  const nomes = ['Luiz', 'Maria', 'João'];
2  console.log(nomes);
3
4  nomes.push('Otávio');
5  console.log(nomes); // Adiciona um item no final
6
7  nomes.unshift('Luiza');
8  console.log(nomes); // Adiciona um item no começo
9
10 nomes.pop();
11 console.log(nomes); // Tira um item do final
12
13 nomes.shift();
14 console.log(nomes); // Tira um item do começo
15
16 delete nomes[1];
17 console.log(nomes); // Tira o item referente ao índice passado, mas sem
    mudar o array
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
[ 'Luiz', 'Maria', 'João' ]
[ 'Luiz', 'Maria', 'João', 'Otávio' ]
[ 'Luiza', 'Luiz', 'Maria', 'João', 'Otávio' ]
[ 'Luiza', 'Luiz', 'Maria', 'João' ]
[ 'Luiz', 'Maria', 'João' ]
[ 'Luiz', <1 empty item>, 'João' ]
```

```
[Done] exited with code=0 in 0.56 seconds
```

Também é possível verificar alguns atributos do array

```
1  const nomes = ['Luiz', 'Maria', 'João'];
2  console.log(nomes);
3
4  console.log(nomes.length); // Mostra o tamanho do array
5
6  console.log(typeof nomes); // Array é do tipo Objeto
7
8  console.log(nomes instanceof Array); // É possível verificar se uma
    variável é um array
```



```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
[ 'Luiz', 'Maria', 'João' ]
3
object
true

[Done] exited with code=0 in 0.592 seconds
```

Para manipular os arrays tem que se ter alguns cuidados para que uma array não interfira no outro

```
1  var nomes = ['Luiz', 'Maria', 'João'];
2  var novos_nomes = nomes
3
4  novos_nomes.pop();
5  console.log(nomes);
6  // Como "novos_nomes" e "nomes" foram referenciados diretamente, quando
7  // fazemos uma operação com um deles, o outro se modifica junto
8
9  var nomes = ['Luiz', 'Maria', 'João'];
10 var novos_nomes = [...nomes]; // Spread server para pegarmos os valores do
    array
11
12 novos_nomes.pop();
13 console.log(nomes);
14 console.log(novos_nomes);
15 // Dessa conseguimos copiar os itens do array para outro
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
[ 'Luiz', 'Maria' ]
[ 'Luiz', 'Maria', 'João' ]
[ 'Luiz', 'Maria' ]

[Done] exited with code=0 in 0.596 seconds
```

A função **slice** serve para separar o array em partes

```
1  const nomes = ['Luiz', 'Maria', 'João', 'Wallace', 'Rosana'];
2
3  var novos_nomes = nomes.slice(0, 1);
4  console.log(novos_nomes); // Vai mostrar apenas primeiro elemento
5
6  var novos_nomes = nomes.slice(0, -2);
7  console.log(novos_nomes); // Vai mostrar todos elementos, menos os dois
    últimos
8
9  var novos_nomes = nomes.slice(3);
10 console.log(novos_nomes); // Vai mostrar todos elementos, menos os três
    primeiros
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
[ 'Luiz' ]
[ 'Luiz', 'Maria', 'João' ]
[ 'Wallace', 'Rosana' ]

[Done] exited with code=0 in 0.573 seconds
```

A função **split** serve para transformar uma string em array

```
1 const nomes = 'Luiz, Maria, João, Wallace, Rosana';
2 const novos_nomes = nomes.split(', ');
3
4 console.log(nomes);
5 console.log(novos_nomes);
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
Luiz, Maria, João, Wallace, Rosana
[ 'Luiz', 'Maria', 'João', 'Wallace', 'Rosana' ]

[Done] exited with code=0 in 0.565 seconds
```

A função **join** serve para transformar uma array em string

```
1 const nomes = ['Luiz', 'Maria', 'João', 'Wallace', 'Rosana'];
2 const novos_nomes = nomes.join(', ');
3
4 console.log(nomes);
5 console.log(novos_nomes);
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
[ 'Luiz', 'Maria', 'João', 'Wallace', 'Rosana' ]
Luiz, Maria, João, Wallace, Rosana

[Done] exited with code=0 in 0.553 seconds
```

A função **splice** é uma junção da função **slice** com as funções de adicionar e remover itens da array

```
1 //           -5      -4      -3      -2      -1
2 //           0       1       2       3       4
3 var nomes = ['Luiz', 'Maria', 'João', 'Wallace', 'Rosana'];
4
5 // (array).splice(índice para começar, número de deletes, elemento para
6 // ser adicionado - 1, elemento para ser adicionado - 2, ...)
7
7 var nomes = ['Luiz', 'Maria', 'João', 'Wallace', 'Rosana'];
8 var novos_nomes = nomes.splice(2);
9 console.log(novos_nomes); // Retorna todos elementos depois do índice 2
```

```

10
11
12 var nomes = ['Luiz', 'Maria', 'João', 'Wallace', 'Rosana'];
13 var novos_nomes = nomes.splice(3, 2);
14 console.log(novos_nomes); // Retorna os elementos que foram removidos
15
16
17 var nomes = ['Luiz', 'Maria', 'João', 'Wallace', 'Rosana'];
18 var novos_nomes = nomes.splice(-4, 1);
19 console.log(novos_nomes); // Retorna os elementos que foram removidos
20
21
22 var nomes = ['Luiz', 'Maria', 'João', 'Wallace', 'Rosana'];
23 var novos_nomes = nomes.splice(2, 3, "Thiago", "José");
24 console.log(nomes); // A array original também foi afetada nesse exemplo
25 console.log(novos_nomes); // Retorna os elementos que foram removidos

```

```

[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
[ 'João', 'Wallace', 'Rosana' ]
[ 'Wallace', 'Rosana' ]
[ 'Maria' ]
[ 'Luiz', 'Maria', 'Thiago', 'José' ]
[ 'João', 'Wallace', 'Rosana' ]

[Done] exited with code=0 in 0.547 seconds

```

A função **concat** serve para concatenar array

```

1  const nums1 = [1, 2, 3];
2  const nums2 = [4, 5, 6];
3
4  var nums3 = nums1 + nums2;
5  console.log(nums3, typeof(nums3)); // Dessa forma as array vão ser
   transformadas em strings e colocadas lado a lado
6
7  var nums3 = nums1.concat(nums2);
8  console.log(nums3, typeof(nums3)); // Dessa forma as array vão ser
   concatenadas e juntadas
9
10 var nova_array = nums3.concat([7, 8, 9], "Nova array concatenada");
11 console.log(nova_array);
12
13 // Tambem seria possivel fazer isso usando spread
14
15 var nova_array = ["Nova array concatenada", ...nums3, ...[7, 8, 9]];
16 console.log(nova_array);

```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
1,2,34,5,6 string
[ 1, 2, 3, 4, 5, 6 ] object
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 'Nova array concatenada' ]
[ 'Nova array concatenada', 1, 2, 3, 4, 5, 6, 7, 8, 9 ]

[Done] exited with code=0 in 0.539 seconds
```

A função **filter** serve para filtrar o array

```
1  const nums = [5, 50, 80, 1, 2, 3, 5, 8, 7, 11, 15, 22, 27];
2
3  var nums_filter = nums.filter(function(valor){
4      return valor >= 10;
5  }); // filter passa os valores, índices e a array (Nessa ordem), dessa
      forma podemos passar uma função para o filter ou criar uma função que só
      funciona nesse filtro
6  console.log(nums_filter);
7
8  var nums_filter = nums.filter((valor, indice, array) =>
9      console.log(indice, valor, array[indice]));
      // Como a função que criamos dentro do filter não precisa de nome podemos
      fazer ela no estilo de arrow function
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
[ 50, 80, 11, 15, 22, 27 ]
0 5 5
1 50 50
2 80 80
3 1 1
4 2 2
5 3 3
6 5 5
7 8 8
8 7 7
9 11 11
10 15 15
11 22 22
12 27 27

[Done] exited with code=0 in 0.795 seconds
```

A função **map** serve para criar um novo array com valores alterados de um outro array

```
1  const nums = [5, 50, 80, 1, 2, 3, 5, 8, 7, 11, 15, 22, 27];
2
3  const numsVezes3 = nums.map(valor => valor*3);
```

```
4 console.log(numsVezes3); // Assim como no filter, a função map passa os valores, índices e a array
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
[
  15, 150, 240, 3, 6, 9,
  15, 24, 21, 33, 45, 66,
  81
]
```

```
[Done] exited with code=0 in 0.563 seconds
```

A função **reduce** serve para reduzir o array a um elemento

```
1 const nums = [5, 50, 80, 1, 2, 3, 5, 8, 7, 11, 15, 22, 27];
2
3 const total = nums.reduce((acumulador, valor) => {
4   return acumulador + valor
5 });
6 console.log(total); // Diferente do filter e do map, o reduce passa o acumulador, os valores, índices e a array (Nessa ordem)
7
8 const lista_pares = nums.reduce((acumulador, valor) => {
9   if (valor%2 === 0) acumulador.push(valor);
10  return acumulador;
11 }, [])
12 console.log(lista_pares); // Também é possível passar um valor inicial para o acumulador
13
14 const erro_acumulador = nums.reduce((acumulador) => {
15   console.log(acumulador);
16 }, 2412); // Lembrando que sempre é preciso dar return no acumulador, caso contrario ele ficara undefined
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tmpCodeRunnerFile.javascript"  
236  
[ 50, 80, 2, 8, 22 ]  
2412  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
undefined  
  
[Done] exited with code=0 in 0.577 seconds
```

O **forEach** é um outro modo de percorrer uma array

```
1  const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9];
2
3  nums.forEach((valor, indice, array) => {
4      console.log(indice, valor, array[indice]);
5  });
```

```
[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"
0 1 1
1 2 2
2 3 3
3 4 4
4 5 5
5 6 6
6 7 7
7 8 8
8 9 9

[Done] exited with code=0 in 0.588 seconds
```

22 Exercícios

Exercício 1

```
1  const nums = [5, 50, 80, 1, 2, 3, 5, 8, 7, 11, 15, 22, 27];
2
3  // Tire os 3 primeiros números e adicione uma array com 5 números no lugar
4  // -- splice e spread
5  // Filtre os números divisíveis por 3 -- filter
6  // Eleve ao quadrado todos os números da array -- map
7  // Faça a soma de todos menos 200 -- reduce
```

Exercício 2

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Document</title>
8  </head>
9  <body>
10   <script src="script.js"></script>
11 </body>
12 </html>
13
14 // Crie um documento js
15 // Receba 10 números que vão ser divididos em duas arrays, onde a primeira
16 // vai conter os números pares e a segunda os impares
17 // Tire o primeiro item da array dos pares e coloque no final da array dos
18 // impares
19 // Coloque em uma terceira array os elementos que são divisíveis por seu
20 // índice
```

23 Correção

```

1  const nums = [5, 50, 80, 1, 2, 3, 5, 8, 7, 11, 15, 22, 27];
2
3  nums.splice(0, 3, ...[1, 2, 3, 4, 5]);
4  console.log(nums);
5
6  const numsFiltrados = nums.filter(value => value%3 === 0);
7  console.log(numsFiltrados);
8
9  const numsAoQuadrado = numsFiltrados.map(value => value**2);
10 console.log(numsAoQuadrado);
11
12 const numFinal = numsAoQuadrado.reduce((soma, value) => {
13   return soma += value;
14 }, -200);
15 console.log(numFinal);

```

[Running] node "C:\Users\liq1ct\AppData\Local\Temp\tempCodeRunnerFile.javascript"

```

[
  1,  2,  3, 4, 5,  1,
  2,  3,  5, 8, 7, 11,
 15, 22, 27
]
[ 3, 3, 15, 27 ]
[ 9, 9, 225, 729 ]
772

```

[Done] exited with code=0 in 0.56 seconds

```

1  var impares = [], pares = [], array = [];
2
3  for (let i = 0; i < 10; i++){
4    let num = prompt("Digite um número:");
5    if (num%2 === 0)
6      pares.push(num);
7    else
8      impares.push(num);
9  }
10
11 impares.push(pares.shift());
12
13 impares.forEach((valor, indice) => {
14   if(valor%indice === 0)
15     array.push(valor);
16 });
17 pares.forEach((valor, indice) => {
18   if(valor%indice === 0)
19     array.push(valor);
20 });

```



```
21  
22 console.log("Pares: " + pares + "\nImpares: " + impares + "\nArray: " +  
    array);
```

Pares: 452,36,4,52,18

Impares: 85,15,95,63,74

Array: 15,63,36,4



24 DOM

A tela da página web é dividida em objetos, tendo como primeiro objeto o **window**, avançando mais teríamos o **document** que é onde começa o arquivo HTML

```
> window
< Window {window: Window, self: Window, document: document, name: '', location: Location, ...}

> window.document
< #document
  <!DOCTYPE html>
  <html lang="en">
    <head>...</head>
    <body> </body>
  </html>
```

Algumas funções básicas para poder começar a modificar o HTML

HTML

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link href="style.css" type="text/css" rel="stylesheet">
8   <title>Document</title>
9 </head>
10 <body>
11   <div class="container">
12     <h1>Algum Texto</h1>
13   </div>
14   <script src="script.js"></script>
15 </body>
16 </html>
```

Java Script

```
1 const container = document.querySelector('.container'); // retorna o
  primeiro elemento relacionado com o que foi passado
2
3 const titulo = container.querySelector('h1');
4 titulo.innerText = "Data"; // Modifica o texto
5
6 const data = new Date();
7 texto = document.createElement('p'); // Cria um novo elemento
8 texto.innerText = data.toLocaleDateString('pt-BR');
9
```

```
10 container.appendChild(texto); // Adciona o elemento na tag
```

Nesse exemplo é possível ver que no HTML o texto estava diferente e não há a tag p junto da data e hora atual.

Também é possível ir no console do site e procurar pelas variáveis criadas no código, que assim será mostrado qual é a parte que aquela variável se refere

```
> container
< ▶ <div class="container">...</div>
> titulo
< <h1>Data</h1>
> data
< Thu Jun 09 2022 11:31:00 GMT-0300 (Horário Padrão de Brasília)
> texto
< <p>09/06/2022</p>
>
```