

JavaScript básico





<title>

AGENDA DO </> </> </> MÓDULO

</title>



Agenda do módulo



Anotações



{Estrutura FOR}

{Estrutura WHILE}

{Utilizando Break}

{Trabalhando com arrays}

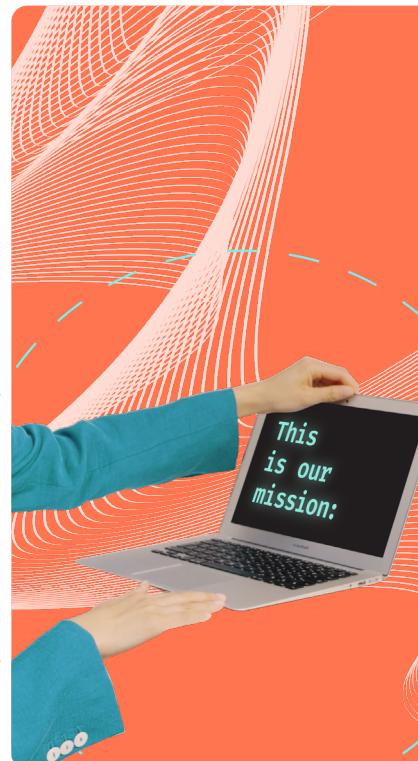


<title>

MISSÃO DO MÓDULO

</title>

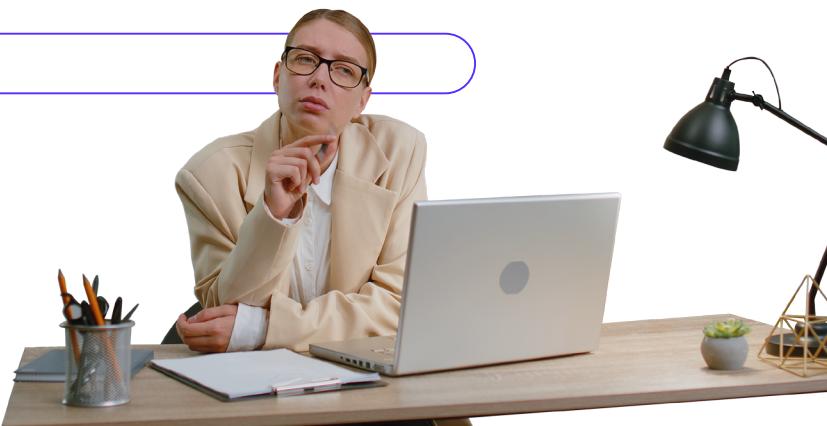
Dar ao código de JavaScript o poder de decisão usando **laços de repetição**.



- Um bom programador é um bom pesquisador.



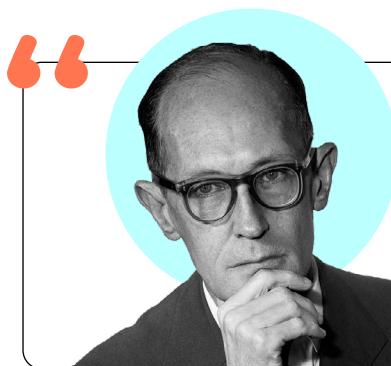
|





{01.}

Estrutura FOR



Perder tempo em aprender coisas que não interessam, priva-nos de descobrir coisas interessantes.

Carlos Drummond de Andrade

Vamos começar falando da estrutura de repetição “for”, que em português, significa “para”.

O “for” nos permite criar loopings em nosso código, ou seja, executar tarefas repetitivas, de maneira dinâmica. Olha só esse exemplo:

```
for (let i = 0; i < 10; i++)  
{  
  console.log(i);  
}
```



Vamos analisar a sintaxe do método “for”:

Podemos definir as declarações em:

O1 Uma ação a ser realizada antes da execução do bloco

O2 A condição para que a execução continue sendo realizada

O3 Uma ação a ser realizada a cada execução do bloco

Mas tenho algo muito importante para falar também, o que chamamos de iterações.

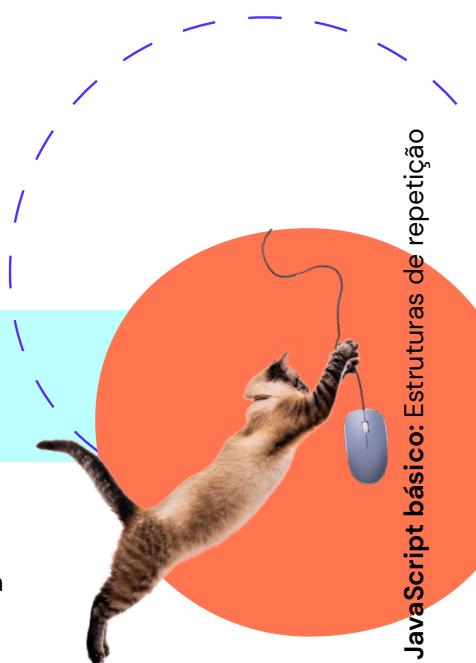
```
<title>
```

PULO DO GATO

```
</title>
```

- Chamamos de “iterações” as vezes em que o looping é executado.

Agora que você já viu como o laço “for” funciona, vamos ver como utilizar a estrutura “for...in”.





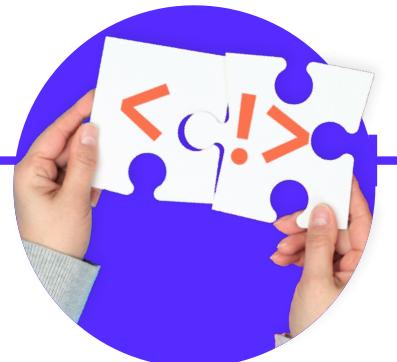
<title>

MÃO + + NA MASSA

</title>

Imagine que você tem um objeto que representa uma lista de pessoas com as suas respectivas idades:

```
const ages = {  
  Robert: 23,  
  Camila: 21,  
  Alfredo: 45,  
  Luana: 12,  
  Denilson: 68,  
};
```



Se tivesse que imprimir no console somente as pessoas com uma idade de número par, como você faria?

Poderíamos usar nesse mão na massa, a estrutura de repetição “for...in” que nos possibilita percorrer por cada pessoa dessa lista e realizar um cálculo para cada idade, assim verificando se é par ou ímpar.

Mas lembre-se que a estrutura “for...in” fornece acesso ao índice do objeto e não ao seu valor.



```
const ages = {  
    Robert: 23,  
    Camila: 21,  
    Alfredo: 45,  
    Luana: 12,  
    Denilson: 68,  
};  
  
for (let person in ages) {  
    const isEven = ages[person] % 2 === 0;  
    if (isEven) {console.log(person);  
}  
}
```

Para cada pessoa da nossa lista de idades, verificamos se a sua idade é módulo de dois. Se sim, imprimimos o nome no console. Nós podemos selecionar o valor do objeto passando o índice dentro dos colchetes (“**objeto[índice]**”), ou seja “**ages[person]**”.

Quando usamos “for... in” é uma boa prática declarar a variável dentro do escopo dele, utilizando a keyword “let” ao invés de declará-la globalmente. Assim evitamos comportamentos indesejados.

Você sabe o que é uma Array?

Provavelmente, você sabe que nós podemos armazenar listas em uma estrutura de dados chamada vetor, ou, Array.



No JavaScript podemos representar um Array inserindo dados dentro de colchetes “[]”. Assim:

```
const names = ["John", "Ana", "Viktor"];
```

```
<title>
```

MÃO + NA MASSA

```
</title>
```

Se tivesse que imprimir no console somente os nomes desse array, como você faria?

Para realizar esse mão na massa e imprimir a lista completa dos itens de um array, utilizamos a estrutura “for...of” que nos permite iterar por cada item de um determinado Array.

```
const names = ["John", "Ana", "Viktor"];

for (let name of names) {
  console.log(name);
}

/*
output:
John
Ana
Viktor
*/
```





Observe que a estrutura “for...of” é muito semelhante ao “for...in”, mas aplicada ao uso de Arrays.

Também, ao contrário da estrutura “for...in”, a “for...of” vai percorrer pelos valores, ignorando os índices.

Caso você queira percorrer pelos índices e valores, é possível com a expressão “Array.entries()”

```
const names = ["John", "Ana", "Viktor"];

for (let name of names.entries()) {
    console.log(name);
}

/*
output:

[ 0, 'John' ]
[ 1, 'Ana' ]
[ 2, 'Viktor' ]
*/
```

<title>

MÃO + + + NA MASSA

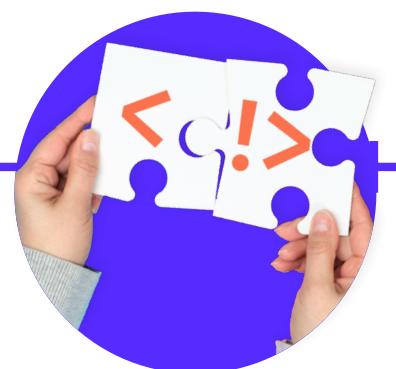
</title>

Dado o array de bebidas:



```
1 const bebidas = ['Refrigerante', 'Água', 'Suco', 'Leite', 'Guaraná'];
```

Utilizando o For ou For of, imprima as palavras contidas no array.

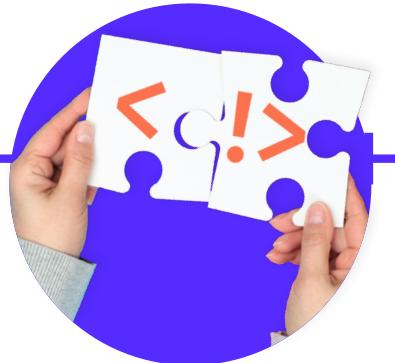




<title>

MÃO + + + NA MASSA

</title>



Dado o array de números:

```
 1 const numbers = [2,4,8,16,32,64,128,256,512,1024,2048];
```

- Utilizando o For ou For of, some todos os valores do array e imprima o resultado.
- Imprima a média aritmética por meio do resultado da soma dos valores (lembrando que a média aritmética é a soma dos valores dividida pela quantidade de elementos).

Dado o objeto:

```
 1 const notas = {  
 2   bimestrel: 7.0,  
 3   bimestre2: 3.7,  
 4   bimestre3: 6.4,  
 5   bimestre4: 8.6  
 6 }
```

- Utilizando o For in, imprima a soma dos valores das notas.
- Após a soma dos valores, descubra a média e faça um if/else para descobrir se o aluno foi aprovado ou não. Com a média sendo 7, o resultado do console, caso seja reprovado, é "O aluno não foi aprovado com a nota...", caso tenha sido aprovado, deverá imprimir "O aluno foi aprovado".



{02.}

Estrutura WHILE

<title>

MÃO + + NA MASSA

</title>



Pesquise sobre o código de JavaScript “while” e encontre as seguintes respostas:

- 1 - O que é **while** em inglês?
- 2 - Qual a função de “**while**” em JS?
- 3 - Qual a diferença entre “**while**” e “**for**”?

Primeiro de tudo, “while” quer dizer “enquanto” em português.

O método “while” é muito parecido com o método “**for**”, mas de forma simplificada.

```
while (condição) {  
    código a ser executado  
}
```



Ao contrário do método “for”, não declaramos uma ação a ser executada antes ou após a execução do bloco. É só uma condição para que ele seja executado.

```
let count = 1;  
while (count <= 5) {  
    console.log(`Estou sendo executado pela ${count} vez. `);  
    count++;  
}
```

<title>

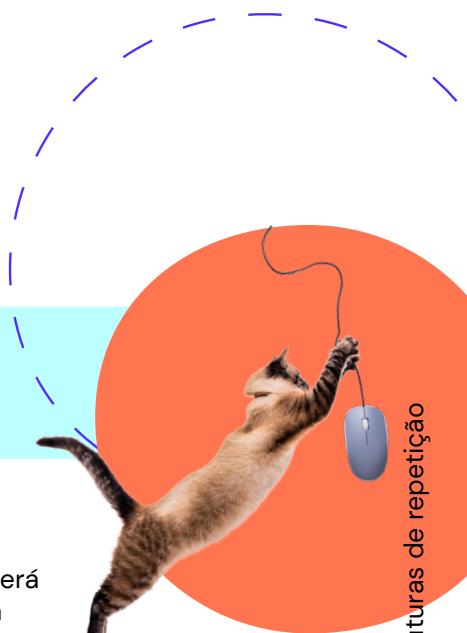
PULO DO GATO

</title>

- Não insira uma condição que sempre será verdadeira.

Observe que a condição fornecida sempre será verdadeira. Isso faz com que o seu programa entre em looping infinito, o que poderá causar um travamento como esse na tela agora. O número 1 sempre será maior do que 0, então essa condição sempre será verdadeira. E é por isso que esse bloco de código será executado infinitamente.

```
while (1 > 0) {  
    console.log("Estou executando infinitamente...");  
}
```





Para evitar isso, devemos inserir em algum lugar do código, algo que altere o valor da nossa condição, como no exemplo abaixo:

```
let i = 0;  
while (i < 1) {  
    console.log("Estou executando somente uma vez.");  
    i++;  
}
```

Observe que nesse exemplo a variável “i” é incrementada a cada iteração.

```
let i = 1;  
while (i <= 10) {  
    console.log(`Estou executando pela ${i} vez.`);  
    i++;  
}
```

Por isso, temos que na décima execução desse looping esse mesmo “i” valerá 10, o que invalida a condição do “while” e ele passa a não ser mais executado.

A estrutura “do...while” é muito semelhante ao “while”, porém, nela vamos indicar **primeiro** o **bloco** a ser executado e **depois** a **condição** para que ele seja executado.

```
do {  
    código a ser executado  
} while (condição);
```



Vamos incrementar a nossa variável “i” a cada execução, enquanto “i” for menor igual a dez.

Note que o exemplo abaixo é exatamente igual ao descrito em “while”, porém organizado de maneira diferente.

```
let i = 1;  
  
do {  
    console.log(`Estou executando pela ${i}  
vez. `);  
    i++;  
} while (i < 10);
```

O uso do...while é interessante em vários casos, como exibir menus, pedir dados ou senhas, por exemplo.

Nesse exemplo abaixo, é possível notar que o looping irá ser executado com certeza um vez, e será repetido enquanto não houver uma inserção de password por parte do usuário.

exemplo.png

```
var password;  
do {  
    password = prompt("Insira sua senha aqui: ")  
} while (!password)
```



A melhor maneira de iniciar é parar de falar e começar a fazer!

Walt Disney

<title>

MÃO + + + NA MASSA

</title>

Crie uma variável **birthYear** que contenha o ano do seu nascimento. Declare também uma variável **currentYear** com o ano atual. Crie duas soluções, uma usando **while** e outra usando **do...while** para imprimir todos os anos, desde o seu nascimento até o ano atual.

Dica: se fizer as duas soluções em um mesmo script, lembre-se de resetar o valor do **birthYear** após a primeira solução, caso necessário.



Caso você tenha ficado com alguma dúvida, acesse o gabarito no gitHub e o passo a passo em vídeo no seu material de apoio.



{03.}

Utilizando Break

Você já viu a keyword “break” na estrutura de controle “switch”, mas você sabia que também é possível usar esse mesmo comando em uma estrutura de repetição como “for” ou “while”?

Imagine a seguinte situação:

Você possui uma sequência numérica, conhecida como Fibonacci.

```
const fib = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34,  
55, 89, 144];
```

E se você quiser imprimir cada número dessa sequência, com exceção daqueles que forem maiores do que 50?

Existem várias soluções para esse problema, mas uma delas é a utilização da keyword “break”. A função do “break” é forçar a interrupção do looping.

```
const fib = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,  
144];  
for (let number of fib) {  
  if (number > 50) {  
    break;  
  }  
  console.log(number);  
}
```



Vamos iterar por cada número desse array e para cada um deles, verificamos se é maior que 50. Caso seja, paramos o looping com o comando “break”.

```
const fib = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144];
for (let number of fib) {
  if (number > 50) {
    break;
  }
  console.log(number);
}
```

O break é um complemento da função de loop usada, seja ela while ou for. Se criamos um loop para encontrar um conjunto de informações, mas só precisamos de parte delas, usamos o break para impedir que o loop continue desnecessariamente depois de encontrarmos a informação que precisávamos.

A keyword “continue” é utilizada dentro dos loopings para evitar a execução do bloco.

```
<title>
```

MÃO + + + NA MASSA

```
</title>
```

Você possui uma lista de clientes armazenados em um array e deseja imprimir um resumo que tenha somente aqueles que possuem crédito maior de 2 mil. Como você faria esse código?





Podemos pensar na seguinte solução para esse mão na massa, utilizando o “continue”.

Observe que logo no início do bloco “for...of”, verificamos se o crédito do cliente é menor do que 2 mil. Caso isso seja verdade, todo o restante do bloco é ignorado e passamos para a próxima iteração. A função do continue é justamente essa: ignorar o código restante do bloco sem parar a execução do looping.

```
const customers = [
  { name: "Ana", age: "37", credit: 4500.0 },
  { name: "Robert", age: "67", credit: 10000.0 },
  { name: "John", age: "18", credit: 1500.0 },
  { name: "Lucas", age: "23", credit: 2300.0 },
  { name: "Maria", age: "20", credit: 2000.0 },
];

for (person of customers) {
  if (person.credit < 2000) continue;

  console.log("-----");
  console.log("Name:", person.name);
  console.log("Age:", person.age);
  console.log("Credit:", person.credit);
}
```



Os grandes erros que se cometem na vida são erros recorrentes. É mesmo essa repetição o que nos define.

Ana Hatherly



<title>

MÃO + + + NA MASSA

</title>

- Crie um laço for para percorrer uma lista que vai de 0 a 20.
- Se os valores forem menores que 5, não deve ser feito nada.
- Caso o valor seja entre 6 e 15, imprima-o no console. Quando a iteração chegar em 16, o laço deve parar.
- Insira as duas condições antes do console.log().

- Caso você tenha ficado com alguma dúvida, acesse o gabarito no github e o passo a passo em vídeo no seu material de apoio.





{04.}

Trabalhando com arrays

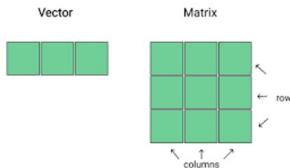
Array é basicamente o encapsulamento de várias informações em uma lista. Com a utilização do array, o programador consegue acessar vários itens que estão dentro da lista de forma individual.

Se arrays não existissem, o armazenamento de vários itens teria que ser feito de forma individual, uma variável para cada item.

O array economiza tempo e otimiza operações por ser menos suscetível a erros.

<title>

PULO DO GATO



- O array possui relação com vetores e matrizes. Vetores são matrizes de uma só dimensão.





Veja esse exemplo.

```
1 const array1 = [1, 2, 3, 4, 5];
2 const array2 = ['Lógica de Programação', 'HTML', 'CSS', 'JS'];
3 const array3 = [[1,1], [1,2], [1,3], [2,1], [2,2], [2,3]];
4
```

O Array 1 e 2 são vetores.

Por outro lado, podemos ter arrays multidimensionais, se tivermos um array dentro do outro. Ou seja, uma matriz.
É o caso do Array 3. Cada item dentro de um par de colchetes é um outro array.

E como adicionamos itens a um array?

Temos duas formas.

O **metodo push**, que serve para adicionar um item no final do array.

```
1 const esportes = ['Basquete', 'Futebol', 'Vôlei'];
2 esportes.push('Xadrez');
3 console.log(esportes);
4 // ['Basquete', 'Futebol', 'Vôlei', 'Xadrez']
```

O outro método é o **unshift** que irá adicionar no início do array.

```
1 const esportes = ['Basquete', 'Futebol', 'Vôlei'];
2 esportes.unshift('Atletismo');
3 console.log(esportes);
4 //['Atletismo', 'Basquete', 'Futebol', 'Vôlei']
```



Existem alguns métodos para remoção de um item no array. Porém, dois métodos são mais comuns e mais utilizados.

O primeiro método seria o **pop** para remover o último item.

```
1 const esportes = ['Basquete', 'Futebol', 'Vôlei'];
2 esportes.pop();
3 console.log(esportes);
4 //['Basquete', 'Futebol']
```

O segundo método é o **shift** que serve para remover o primeiro item.

```
1 const esportes = ['Basquete', 'Futebol', 'Vôlei'];
2 esportes.shift();
3 console.log(esportes);
4 //['Futebol', 'Vôlei']
```

```
<title>
```

MÃO + + + NA MASSA

```
</title>
```

Dado o array de strings:

```
1 const array = ['Lógica de Programação', 'HTML', 'CSS', 'JavaScript', 'Github', 'Git', 'Algoritmos', 'Dados'];
```



Utilize os métodos `pop` e `shift` para remover o primeiro e o último item. Depois adicione um número, utilizando o método `unshift` e o método `push`.

Caso você tenha ficado com alguma dúvida, acesse o gabarito no [gitbub](#) e o passo a passo em vídeo no seu material de apoio.



Um outro ponto muito importante é que a contagem do índice de um array, começa no número 0.

Ou seja, numa lista com 5 itens, os indices são:

0, 1, 2, 3, 4

A verificação do tamanho de um array é muito importante, pois dá a dimensão de quantos valores estão encapsulados dentro daquele array. O que torna sua manutenção ou alteração mais prática, conseguindo localizar alguma informação de forma mais direta.

```
1 const esportes = ['Basquete', 'Futebol', 'Vôlei'];
2 console.log(esportes.length);
3 //3
```

<title>

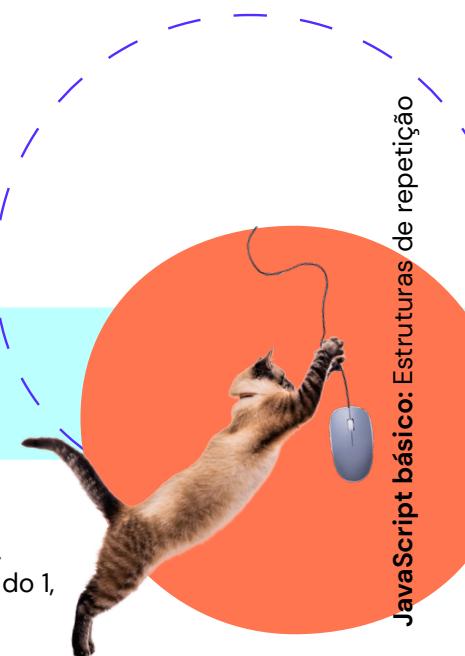
PULO DO GATO

</title>

- O índice não é a mesma coisa que o tamanho do array.

O que isso quer dizer?

O índice começa sua contagem no número 0, como já visto. Já o tamanho se conta a partir do 1, pois está contando os valores e não o índice.





Mas antes de sair alterando um array, às vezes será necessário confirmar que uma determinada variável é um array. A partir da versão ES5 é possível fazer essa verificação usando o `Array.isArray(variável)`.

```
let arr = [1,2,3,4,5];
if (Array.isArray(arr)) {
  console.log("É um array!");
}
```

No caso de um array unidimensional, ou seja, possuindo apenas itens que não sejam arrays, você pode acessar dessa forma.

```
const array1 = [1, 2, 3, 4, 5];
console.log(array1[1]);
//2
```

Já no caso de um array multidimensional, em que há arrays dentro do array, o acesso é feito de forma similar, porém o desenvolvedor vai **primeiro localizar a posição** que o array se encontra e depois, dentro deste array encontrado, **qual posição deseja encontrar**.

Como podemos observar, há seis arrays dentro de um array maior.

```
const array3 = [[1,2], [3,4], [5,6], [7,8], [9,10], [11,12]];
console.log(array3[3][1]);
//8
```

Se quissemos encontrar o valor 8, como faríamos?

No caso, eu estou querendo localizar o array que está na posição 3, lembrando que o índice do array começa no 0. Logo, o array que eu encontrei é o array [7,8].

Como queremos o valor que está no índice [1] dentro desse array, encontraremos o valor 8.



<title>

MÃO + NA MASSA

</title>

- Dado o array multidimensional:
- Indique os valores dos seguintes índices: [2][4], [1][7], [0][8], [2][6] e [1][0].

Caso você tenha ficado com alguma dúvida, acesse o gabarito no github e o passo a passo em vídeo no seu material de apoio.



```
1 const matriz = [
2   [
3     11, 12, 13,
4     21, 22, 23,
5     31, 32, 33
6   ],
7   [
8     41, 42, 43,
9     51, 52, 53,
10    61, 62, 63
11  ],
12  [
13    71, 72, 73,
14    81, 82, 83,
15    91, 92, 93
16  ]
17];
```

Quando há a construção de um array, é comum que possa ocorrer erros de digitação ou apenas uma necessidade de precisar trocar algum valor dentro do array em algum momento. Para isso, a linguagem também possui recursos para corrigir esses erros sem precisar ficar procurando.



```
1 const esportes = ['Basquete', 'Futebol', 'Vôlei'];
2 esportes[0] = 'Atletismo'
3 console.log(esportes);
4 //['Atletismo', 'Futebol', 'Vôlei']
```

E já ouviu falar no método sort?

Através dele o programador tem a facilidade de conseguir ordenar da forma que achar melhor. Para ordenar de maneira simples, basta adicionar o método no array em questão. Olha esses exemplos de ordenação tanto com números quanto com strings.

```
1 const numeros = [23,30,43,77,24,11,33,34];
2 numeros.sort();
3 console.log(numeros);
4 // [11, 23, 24, 30, 33, 34, 43, 77]
```

```
1 const esportes = ['Basquete', 'Atletismo', 'Taekwondo', 'Judô', 'Vôlei', 'Futebol'];
2 esportes.sort();
3 console.log(esportes);
4 //['Atletismo', 'Basquete', 'Futebol', 'Judô', 'Taekwondo', 'Vôlei']
```

Há diversos tipos de manipulação de array. Uma delas é iterar itens através de estruturas de repetição.



O exemplo abaixo mostra como iterar valores através do laço de repetição FOR.

```
1 const array1 = [1, 2, 3, 4, 5];
2 for (let i = 0; i < array1.length; i++) {
3   console.log(array1[i]);
4 }
5 //1
6 //2
7 //3
8 //4
9 //5
```

<title>

MÃO + NA MASSA

</title>

- Dado o array de strings:

```
1 const array = ['Lógica de Programação', 'HTML', 'CSS', 'JavaScript', 'Github', 'Git', 'Algoritmos', 'Dados'];
```

- Ordene o array atribuindo este para uma nova constante. Após a ordenação, verifique o tamanho do array e indique qual string está localizado no índice 6.

Caso você tenha ficado com alguma dúvida, acesse o gabarito no gitbook e o passo a passo em vídeo no seu material de apoio.





+

+

+

+

+

+

Se um homem começar com certezas, ele deverá terminar em dúvidas; mas se ele se satisfizer em começar com dúvidas, deverá terminar em certezas.

Francis Bacon

<title>

DESAFIO CONQUER

</title>



if (challenge)
// start

<body>

- > Calcule a média dos números de um array de números.
- > Faça de duas maneiras: uma usando um laço de repetição **for** e outra usando um laço de repetição **while**.
- > Imprima/apresente o resultado em tela.

</body>



<title>

QUERO MAIS

</title>

👉 Loopings e iterações em JavaScript

👉 Métodos de manipulação de arrays

👉 Métodos de manipulação de objetos

👉 Recursividade em JavaScript (tópico avançado)

👉 Exercícios extra





Anotações

-
-
-
-
-
-

010101
0001001100
0100010000001010
0101000101010010
000010001001010100010
10100001010101000100010
100010100111001011100101
111100101100001011001010101001000001010010001100100000010100