Mais métodos de arrays

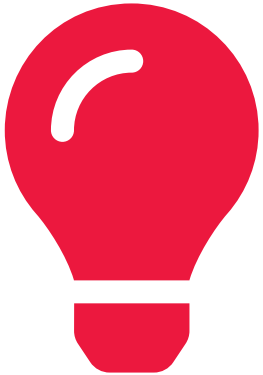
Certified Developer
The Ultimate Tech Degree

DigitalHouse >
Coding School

Temas

1. .slice()
2. .splice()
3. .sort()
4. .flat()
5. .includes()
6. .find()

Se algo é muito usado em programação, as linguagens costumam nos dar uma **versão abreviada**.



1 | .slice()

.slice()

Este método retorna **uma cópia do array**. Para isso é necessário determinar dois parâmetros, índice **inicial** e **final** (opcional).

Caso o índice inicial seja negativo, é extraído os elementos se iniciando no final do array, caso omitido o **padrão** é 0 e se for um índice maior que o tamanho do array, **retornará** um array **vazio**.

Caso o índice final seja negativo, ele conta para o final, `array.slice(3, -1)` extrai do quarto elemento até o penúltimo. **Se** for **maior que** o **tamanho** do array ou omitido, **extrai** para o **final** do array.

```
{}
array.slice(inicio, fim);
// indicamos os índices de inicio e fim
// para obter um novo array
```

{código}

```
let numeros = [3, 4, 5, 6, 7];

let subArray = numeros.slice(0, 3);

console.log(subArray); // [3,4,5]
```

Declaramos a variável **numeros** e atribuímos alguns valores.

{código}

```
let numeros = [3, 4, 5, 6, 7];

let subArray = numeros.slice(0, 3);

console.log(subArray); // [3,4,5]
```

Para **.slice()** definimos o índice **inicial** e **final**, o elemento do índice final não estará no **subArrray**.

A variável **subArray** conterá o novo array criada a partir do array original **números**.

{código}

```
let numeros = [3, 4, 5, 6, 7];

let subArray = numeros.slice(0, 3);

console.log(subArray); // [3,4,5]
```

Mostra os valores do novo **subArray**, que contem os elementos do array **numeros**, mas com índices limitados.

2 | .splice()

.splice()

Este método nos ajuda a **remover** ou **adicionar** elementos de um array.

Recebe três parâmetros:
Inicio: O índice do primeiro elemento (onde começará a mudança).

Quantidade: Opcional - Número de inteiros a eliminar (deve ser do tipo **inteiro**).
Itens: Indica os elementos que serão adicionados ao array. Caso seja omitido, ele apenas remove.

```
{}
array.splice(inicio, quantidade, item1, item2, ...);
// Indicamos o inicio, itens a serem removidos
// e os itens que serão adicionados.
```

{código}

```
let numeros = [3, 4, 5, 6, 7];

numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

Declaramos a variável **numeros** e atribuímos alguns valores.

{código}

```
let numeros = [3, 4, 5, 6, 7];

numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

Aplicamos a função **.splice()** passando como **inicio** o índice 0, **elimina** 0 elementos e **adiciona** o valor 2.

{código}

```
let numeros = [3, 4, 5, 6, 7];

numeros.splice(0, 0, 2);
console.log(numeros); // [2,3,4,5,6,7]

numeros.splice(1, 2);
console.log(numeros); // [2,5,6,7]
```

Neste caso, iremos começar a partir do índice 1, e iremos eliminar 2 (dois) elementos a partir dele.

3 | .sort()

.sort()

Este método nos ajuda a ordenar os elementos de um array.

Ele recebe um callback como um parâmetro (opcional) que especifica o modo de classificação, se for omitido, o array é ordenado com o valor da string (Unicode), converte cada elemento em uma string.

A callback recebe dois parâmetros, ambos elementos que serão comparados.

```
{}
array.sort(); // ordenar com posição de valor Unicode

array.sort(callback); // a callback com parâmetros
```

{código}

```
let marcas = ['samsung', 'xiaomi', 'apple'];

marcas.sort();
// ['apple', 'samsung', 'xiaomi']
```

Declaramos a variável **marcas** e atribuímos alguns valores.

{código}

```
let marcas = ['samsung', 'xiaomi', 'apple'];
```

```
marcas.sort();
// ['apple', 'samsung', 'xiaomi']
```

Aplicamos a ordenação através do método **.sort()** e obtemos o resultado.

{código}

```
let numeros = [10, 3, 4, 52, 6, 7];
```

Declaramos a variável **numeros** e atribuímos alguns valores.

```
numeros.sort(
  function(a,b){
    return a-b;
}); // [4, 7, 8, 9, 10, 52]
```

{código}

```
let numeros = [10, 3, 4, 52, 6, 7];

numeros.sort(
  function(a,b){
    return a-b;
}); // [4, 7, 8, 9, 10, 52]
```

Aplicamos o método **.sort()**, passando uma callback como parâmetro.

{código}

```
let numeros = [10, 3, 4, 52, 6, 7];

numeros.sort(
  function(a,b){
    return a-b;
}); // [4, 7, 8, 9, 10, 52]
```

A callback por sua vez, recebe dois parâmetros. O número e anterior que será comparado.

{código}

```
let numeros = [10, 3, 4, 52, 6, 7];

numeros.sort(
  function(a,b){
    return a-b;
}); // [4, 7, 8, 9, 10, 52]
```

Definimos uma lógica para ordenar do menor para o maior.

4 | .flat()

.flat()

Este método é usado para desembrulhar sub-arrays dentro de arrays, ou seja, ele não modifica o array original.

Recebe como parâmetro (opcional) um nível de profundidade que especifica o quanto o array aninhado deve ser achatado, se for omitido, por padrão leva 1 como valor.

```
{}
array.flat(); // parâmetro omitido

array.flat(n); // parâmetro definido. Nível n.
```

{código}

```
let numeros = [1, 2, 3, [4], [[5, 6]]];
```

Declaramos a variável **numeros** e atribuímos alguns valores.

```
let novoArray = numeros.flat();
// [1, 2, 3, 4, [[5, 6]]

novoArray = numeros.flat(2);
// [1, 2, 3, 4, 5, 6]
```

{código}

```
let numeros = [1, 2, 3, [4], [[5, 6]]];

let novoArray = numeros.flat();
// [1, 2, 3, 4, [[5, 6]]

novoArray = numeros.flat(2);
// [1, 2, 3, 4, 5, 6]
```

Aplicamos o método **.flat()** e omitimos o nível, que passou a ser 1.

{código}

```
let numeros = [1, 2, 3, [4], [[5, 6]]];

let novoArray = numeros.flat();
// [1, 2, 3, 4, [[5, 6]]

novoArray = numeros.flat(2);
// [1, 2, 3, 4, 5, 6]
```

Aplicamos o método **.flat()** e definimos o nível 2.

5 | .includes()

.includes()

Verifica a existência de um elemento no array, retornando **true** caso exista, e **false** se não.

Parâmetros que recebe:
Elemento: valor que será procurado no array.
Inicio: Índice na qual se iniciará a busca.

```
{}
array.includes(elemento, inicio);
// indicamos o elemento a buscar
// e a posição que começará a busca
```

{código}

```
let numeros = [3, 4, 5, 6, 7];

let existe = numeros.includes(4);
// true

existe = numeros.includes(4, 2);
// false
```

Declaramos a variável **numeros** e atribuímos alguns valores.

{código}

```
let numeros = [3, 4, 5, 6, 7];

let existe = numeros.includes(4); // true

existe = numeros.includes(4, 2); // false
```

Aplicamos o método **.includes()** e passamos como parâmetro o **número 4**.

Nesse momento ele **busca** pela **existência** do valor 4 dentro do array.

Como encontra, retorna **true.**

{código}

```
let numeros = [3, 4, 5, 6, 7];

let existe = numeros.includes(4); // true

existe = numeros.includes(4, 2); // false
```

Aplicamos o método **.includes()** e passamos como parâmetro o **número 4 e 2**.

Nesse momento ele **busca** pela **existência** do valor 4 dentro do array, a partir do 2 índice.

Como o valor não existe após o 2 índice, retorna **false.**

6 | .find()

.find()

Retorna o primeiro valor que cumprir a condição especificada na callback.
A callback irá retornar **true** ou **false**, e receberá três parâmetros:

Elemento: elemento atual do array.
Indice: Opcional – posição atual do elemento.
Array: array que está sendo percorrido.

```
{}  array.find(callback(elemento, indice, array));
    // indicamos o elemento a buscar
    // e a posição que irá começar
```

{código}

```
let moedas = [
  { nome: 'Bitcoin', simbolo: 'BTC' },
  { nome: 'Bitcoin', simbolo: 'BTC' },
  { nome: 'Ethereum', simbolo: 'ETH' },
  { nome: 'Cardano', simbolo: 'ADA' },
];
```

Declaramos o array **moedas** que irá armazenar diversos **objetos**.

```
moedas.find(
  function (moeda) {

    return moeda.nome === 'Bitcoin';

}); // {nome: 'Bitcoin', simbolo: 'BTC'}
```

{código}

```
let moedas = [
  { nome: 'Bitcoin', simbolo: 'BTC' },
  { nome: 'Bitcoin', simbolo: 'BTC' },
  { nome: 'Ethereum', simbolo: 'ETH' },
  { nome: 'Cardano', simbolo: 'ADA' },
];

moedas.find(
  function (moeda) {
    return moeda.nome === 'Bitcoin';
}); // {nome: 'Bitcoin', simbolo: 'BTC'}
```

Aplicamos o método **.find()** ao array **moedas**, passando como parâmetro uma **callback**.

{código}

```
let moedas = [
  { nome: 'Bitcoin', simbolo: 'BTC' },
  { nome: 'Bitcoin', simbolo: 'BTC' },
  { nome: 'Ethereum', simbolo: 'ETH' },
  { nome: 'Cardano', simbolo: 'ADA' },
];

moedas.find(
  function (moeda) {
    return moeda.nome === 'Bitcoin';
}); // {nome: 'Bitcoin', simbolo: 'BTC'}
```

A **callback** receberá um parâmetro, que será o elemento do array percorrido.

{código}

```
let moedas = [
  { nome: 'Bitcoin', simbolo: 'BTC' },
  { nome: 'Bitcoin', simbolo: 'BTC' },
  { nome: 'Ethereum', simbolo: 'ETH' },
  { nome: 'Cardano', simbolo: 'ADA' },
];

moedas.find(
  function (moeda) {
    return moeda.nome === 'Bitcoin';
}); // {nome: 'Bitcoin', simbolo: 'BTC'}
```

Com base em uma **condição**, dizemos qual elemento queremos retornar, devolvendo **true** ou **false** pela **callback.**


DigitalHouse>
Coding School