

Manipulação Array ([JavaScript Playground \(playcode.io\)](https://playcode.io))

Neste curso você aprenderá a manipular um array utilizando as seguintes funções nativas:

- push
- splice
- pop
- shift
- forEach
- map
- filter
- reduce
- sort

Funções nativas de array

O que são?

É comum utilizarmos um array para armazenar dados. Por exemplo:

```
const diasDaSemana = [  
  "domingo",  
  "segunda-feira",  
  "terça-feira",  
  "quarta-feira",  
  "quinta-feira",  
  "sexta-feira"  
];
```

Armazenando os dias da semana.

Para manipular este array contamos com algumas funções como, por exemplo:

```
const diasDaSemana = [  
  "domingo",  
  "segunda-feira",  
  "terça-feira",  
  "quarta-feira",  
  "quinta-feira",  
  "sexta-feira"  
];
```

```
diasDaSemana.push("sábado");
```

```
diasDaSemana.map( (dia)=> console.log(dia) );
```

Função para inserir
um valor no array.

Função para percorrer um array.

Essas funções existem sem você as ter
programado e por isso são chamadas de
funções nativas.

Funções nativas de array

Por que são úteis?

```
const diasDaSemana = [ "domingo", "segunda-feira", "terça-feira",  
diasDaSemana.map( (dia)=> console.log(dia) );
```



Função nativa para percorrer um array

Veja outro exemplo:

Dessa vez vamos adicionar um novo elemento no array.

Sem uma função nativa

```
const diasDaSemana = [ "domingo", "segunda-feira", "terça-feira",  
const total = diasDaSemana.length;  
diasDaSemana[total] = "sábado";
```

Com uma função nativa (**push**)

```
const diasDaSemana = [ "domingo", "segunda-feira", "terça-feira",  
diasDaSemana.push("sábado");
```

Utilizar as funções nativas reduz códigos repetitivos e agiliza o processo de desenvolvimento.

As vantagens do uso das funções nativas (métodos de array) são:

- Redução do código repetitivo.
- Economia de tempo.
- Aceleração no processo de desenvolvimento.

Array

O que é?

Array é uma coleção de dados que pode armazenar mais de um valor em apenas uma variável.

Os valores armazenados podem ser de diversos tipos.

```
const produtos = ["Notebook x40", "iPhone X", "Mouse Microsoft"];
const valores = [ 4010.99, 10000.76, 90.15 ];
const itensVenda = [
  { produto: "Notebook x40", valor: 4010.99, ativo: true },
  { produto: "iPhone X", valor: 10000.76, ativo: true },
  { produto: "Mouse Microsoft", valor: 90.15, ativo: false },
];
```

Array de **string**

Array de **number**

Array de **objetos**

O **Código 1** apresenta exemplos de array.

```
1  const alunos = ['João', 'Diego', 'Eduardo', 'Amanda'];
2
3  const produtos = ["Notebook x40", "iPhone X", "Mouse Microsoft"];
4
5  const valores = [ 4010.99, 10000.76, 90.15 ];
6
7  const itensVenda = [
8    { produto: "Notebook x40", valor: 4010.99, ativo: true },
9    { produto: "iPhone X", valor: 10000.76, ativo: true },
10   { produto: "Mouse Microsoft", valor: 90.15, ativo: false },
11 ];
```

Código 1. Exemplos de array

Já sabemos acessar e alterar um elemento do array. Além disso, também já sabemos quantificar os elementos que ele tem. Na **Figura 1** relembremos como isso é feito.

1. Inserindo e removendo um elemento no array

.push()
Função nativa

Inserir um elemento em um array é bem simples. Basta utilizar o método **push**.

```
const diasDaSemana = [  
  "domingo",  
  "segunda-feira",  
  "terça-feira",  
  "quarta-feira",  
  "quinta-feira",  
  "sexta-feira"  
];  
  
diasDaSemana.push("sábado");
```

Inserindo um
elemento no array

Este novo elemento vai ser inserido no final do array.

```
[  
  'domingo',  
  'segunda-feira',  
  'terça-feira',  
  'quarta-feira',  
  'quinta-feira',  
  'sexta-feira',  
  'sábado'  
]
```

Elemento inserido.

Função nativa

A função splice remove um ou mais elementos de um array.

Removendo o valor "terça-feira"

A função `splice` recebe dois parâmetros

```
diasDaSemana.splice(2,1);
```

A posição inicial

Quantos elementos
serão removidos

Veja no **Código 1** exemplos de remoção de elementos.

```
1  const diasDaSemana = [  
2    "domingo", "segunda-feira", "terça-feira",  
3    "quarta-feira", "quinta-feira", "sexta-feira", "sábado"  
4  ];  
5  
6  diasDaSemana.splice(2,1);  
7  // removendo "terça-feira"  
8  
9  diasDaSemana.splice(1,3);  
10 // removendo "segunda-feira", "terça-feira", "quarta-feira"
```

Código 1. Removendo elementos de um array

Em alguns casos queremos remover o último ou o primeiro elemento de um array. Para isso utilizamos as funções `pop` e `shift`, que podem ser vistas na **Figura 1**.

```
const diasDaSemana = [
  "domingo", "segunda-feira", "terça-feira",
  "quarta-feira", "quinta-feira", "sexta-feira", "sábado"
];
```

```
diasDaSemana.pop();
diasDaSemana.shift();
```

Removendo o
primeiro elemento
"domingo"

Removendo o
último elemento
"sábado"

Exemplo prático

Veja um exemplo prático na **Figura 2** da manipulação de um array utilizando as funções que aprendemos nesta aula.

```
1  const produto = {
2    nome: 'New Super Mario Bros.', qnt: 1, valor: 250
3  };
4
5  const carrinho = [
6    { nome: 'The Legend of Zelda', qnt: 1, valor: 250 },
7    { nome: 'Super Mario Kart 8', qnt: 1, valor: 300 },
8  ];
9
10 // Insere o produto no carrinho
11 carrinho.push(produto);
12
13 // Remove o item "Super Mario Kart 8"
14 carrinho.splice(1,1);
15
16 // Remove todos os elementos do carrinho
17 const totalElementos = carrinho.length;
18 carrinho.splice(0,totalElementos);
```

Veja no **Código 2** o exemplo prático com funções de array.

```
1  const produto = {
2    nome: 'New Super Mario Bros.', qnt: 1, valor: 250
3  };
4
5  const carrinho = [
6    { nome: 'The Legend of Zelda', qnt: 1, valor: 250 },
7    { nome: 'Super Mario Kart 8', qnt: 1, valor: 300 },
8  ];
9
10 // Insere o produto no carrinho
11 carrinho.push(produto);
12
13 // Remove o item "Super Mario Kart 8"
14 carrinho.splice(1,1);
15
16 // Remove todos os elementos do carrinho
17 const totalElementos = carrinho.length;
18 carrinho.splice(0,totalElementos);
```

Código 2. Exemplo prático de funções de array

2. .forEach()

[Voltar](#)

Suporte ao aluno Anotar Marcar como concluído

Uma forma fácil de percorrer um array é utilizando a função nativa `forEach`. Através dela conseguimos executar uma função para cada elemento do array.

Aprenda no **Flow** abaixo a função nativa (método) `forEach`.

.forEach()

Função nativa

A função **forEach** é usada para percorrer um array e executar uma função para cada elemento.

```
const produtos = [  
  { id: 1, nome: 'Açucar', estoque: 100, valor: 2.00 },  
  { id: 2, nome: 'Álcool 70', estoque: 50, valor: 9.95 },  
  { id: 3, nome: 'Luvas descartáveis', estoque: 1000, valor: 2.50 },  
];  
  
function imprimirProduto (produto) {  
  console.log(produto.nome);  
}  
  
produtos.forEach(imprimirProduto);
```

Executando a função **imprimirProduto** para cada elemento do array **produtos**.

```
produtos.forEach(imprimirProduto);
```

A função nativa `forEach` executando a função `imprimirProduto` para cada elemento do array `produtos`.

```
imprimirProduto( produtos[0] );  
imprimirProduto( produtos[1] );  
imprimirProduto( produtos[2] );
```

Nome do
array

Função que será
executada em cada
elemento do array

```
produtos.forEach(imprimirProduto);
```

Função
nativa

No **Flow** abaixo você aprenderá sobre a função que é passada para o `forEach` e os parâmetros recebidos.

`.forEach()`

Função que será executada

A função que será executada recebe como parâmetro o elemento do array que está sendo iterado.

Exemplo prático

O `forEach` pode ser utilizado para percorrer um array a fim de imprimir no terminal os itens de um carrinho de compra, como pode ser visto na **Figura 2**.

```
const carrinho = [
  { nome: 'The Legend of Zelda', qnt: 1, valor: 250 },
  { nome: 'Super Mario Kart 8', qnt: 1, valor: 300 },
  { nome: 'New Super Mario Bros.', qnt: 1, valor: 250 }
];

function imprimirItem( produto, index ) {
  let texto = index + ' - ';
  texto += produto.qnt + ' UN - ';
  texto += produto.nome + ' - ';
  texto += 'R$ ' + produto.valor + ' - ';
  texto += 'R$ ' + produto.qnt * produto.valor;

  console.log( texto );
}

carrinho.forEach(imprimirItem);
```

Veja no **Código 2** o exemplo do `forEach`.

```
1  const carrinho = [
2    { nome: 'The Legend of Zelda', qnt: 1, valor: 250 },
3    { nome: 'Super Mario Kart 8', qnt: 1, valor: 300 },
4    { nome: 'New Super Mario Bros.', qnt: 1, valor: 250 }
5  ];
6
7  function imprimirItem( produto, index ) {
8    let texto = index + ' - ';
9    texto += produto.qnt + ' UN - ';
10   texto += produto.nome + ' - ';
11   texto += 'R$ ' + produto.valor + ' - ';
12   texto += 'R$ ' + produto.qnt * produto.valor;
13
14   console.log( texto );
15 }
16
17 carrinho.forEach(imprimirItem);
```

Código 2. Utilizando `forEach` para imprimir os elementos de um array

3. map()

[Voltar](#)

Suporte ao aluno Anotar Marcar como concluído

A função nativa (método) `map` é muito utilizada quando queremos criar um array a partir de outro. Esse novo array possui a mesma quantidade de elementos, porém o valor do elemento será diferente do original.

O **Flow** abaixo demonstra como `map` é utilizado.

`.map()`
Função nativa

A função `map` é utilizada para percorrer um array e criar um novo com os elementos alterados. Por exemplo:

```
const produtosCadastrados = [  
  { id: 1, nome: 'Açúcar', estoque: 100, valor: 2.00 },  
  { id: 2, nome: 'Álcool 70', estoque: 50, valor: 9.95 },  
  { id: 3, nome: 'Luvas descartáveis', estoque: 1000, valor: 2.50 },  
];
```

```
function retornaProduto(produto) {  
  const produtoExibicao = {  
    nome: produto.nome,  
    valor: produto.valor  
  }  
  
  return produtoExibicao;  
}
```

Executando a função **retornaProduto** para cada elemento do array **produtosCadastrados**.



```
const produtosExibicao = produtosCadastrados.map(retornaProduto);  
  
console.log(produtosExibicao);
```

```
const produtosCadastrados = [  
  { id: 1, nome: 'Açúcar', estoque: 100, valor: 2.00 },  
  { id: 2, nome: 'Álcool 70', estoque: 50, valor: 9.95 },  
  { id: 3, nome: 'Luvas descartáveis', estoque: 1000, valor: 2.50 },  
];
```

```
function retornaProduto(produto) {  
  const produtoExibicao = {  
    nome: produto.nome,  
    valor: produto.valor  
  }  
  
  return produtoExibicao;  
}
```

A função **retornaProduto** retorna um objeto com apenas o **nome** e o **valor** do produto.

```
const produtosExibicao = produtosCadastrados.map(retornaProduto);  
  
console.log(produtosExibicao);
```

Array original com todos os dados do produto.

```
const produtosCadastrados = [
  { id: 1, nome: 'Açúcar', estoque: 100, valor: 2.00 },
  { id: 2, nome: 'Álcool 70', estoque: 50, valor: 9.95 },
  { id: 3, nome: 'Luvas descartáveis', estoque: 1000, valor: 2.50 },
];
```

Novo array (**produtosExibicao**) que só possui o nome e o valor do produto.

```
[
  { nome: 'Açúcar', valor: 2.00 },
  { nome: 'Álcool 70', valor: 9.95 },
  { nome: 'Luvas descartáveis', valor: 2.50 },
];
```

Exemplo 1

Um exemplo do uso da função `map` pode ser visto na **Figura 2**. Criaremos um array que possui uma string com os dados de um carro.

```
const carros = [
  { marca: 'Fiat', modelo: 'Uno', anoFabricacao: 2015 },
  { marca: 'GM', modelo: 'Onix', anoFabricacao: 2018 },
  { marca: 'Ford', modelo: 'KA+', anoFabricacao: 2018 },
  { marca: 'Fiat', modelo: 'Cronos', anoFabricacao: 2020 },
];

function retornaCarro(carro) {
  return carro.marca + ' ' + carro.modelo + ' ano: ' + carro.anoFabricacao;
}

const novosCarros = carros.map(retornaCarro);

console.log(novosCarros);
```

Figura 2. Utilizando `map` para criar um novo array

```
1  const carros = [  
2    { marca: 'Fiat', modelo: 'Uno', anoFabricacao: 2015 },  
3    { marca: 'GM', modelo: 'Onix', anoFabricacao: 2018 },  
4    { marca: 'Ford', modelo: 'KA+', anoFabricacao: 2018 },  
5    { marca: 'Fiat', modelo: 'Cronos', anoFabricacao: 2020 },  
6  ];  
7  
8  function retornaCarro(carro) {  
9    return carro.marca + ' ' + carro.modelo + ' ano: ' + carro.anoFabricacao;  
10 }  
11  
12 const novosCarros = carros.map(retornaCarro);  
13  
14 console.log(novosCarros);  
15  
16 /*  
17  * vai imprimir:  
18  [  
19    'Fiat Uno ano: 2015',  
20    'GM Onix ano: 2018',  
21    'Ford KA+ ano: 2018',  
22    'Fiat Cronos ano: 2020',  
23    'GM Onix ano: 2018',  
24    'Ford KA+ ano: 2018',  
25    'Fiat Cronos ano: 2020'  
26  ]  
27  */
```

Código 1. Exemplo de map

Exemplo 2

Outro exemplo é um array que possui os meses do ano. Utilizamos o `map` para criar um array com apenas as três primeiras letras de cada elemento.

O exemplo 2 pode ser visto na **Figura 4**.

```
const meses = [  
  "Janeiro", "Fevereiro", "Março", "Abril",  
  "Maio", "Junho", "Julho", "Agosto",  
  "Setembro", "Outubro", "Novembro", "Dezembro"  
];  
  
function abreviar(mes) {  
  return mes.substr(0,3)  
}  
  
const mesesAbreviados = meses.map(abreviar);  
  
console.log(mesesAbreviados);
```

Figura 4. Array com os meses abreviados

Veja no **Código 2** o exemplo de visto na **Figura 5**.

```
1  const meses = [  
2    "Janeiro", "Fevereiro", "Março", "Abril",  
3    "Maio", "Junho", "Julho", "Agosto",  
4    "Setembro", "Outubro", "Novembro", "Dezembro"  
5  ];  
6  
7  function abreviar(mes) {  
8    return mes.substr(0,3)  
9  }  
10  
11  const mesesAbreviados = meses.map(abreviar);  
12  
13  console.log(mesesAbreviados);  
14  
15  /**  
16   * Vai imprimir  
17   * [  
18   *   'Jan', 'Fev', 'Mar',  
19   *   'Abr', 'Mai', 'Jun',  
20   *   'Jul', 'Ago', 'Set',  
21   *   'Out', 'Nov', 'Dez'  
22   * ]  
23   */
```

Código 2. Exemplo da Figura 5

4. .filter()

Existem casos em que selecionamos apenas alguns elementos de um array, e para isso utilizamos a função nativa `filter`

A função **filter** é utilizada para filtrar os elementos de um array e criar um novo array com apenas os elementos que atendem à uma condição. Por exemplo:


```
const produtos = [
  { id: 1, nome: 'Açúcar', ativo: true, valor: 2.00 },
  { id: 2, nome: 'Álcool 70', ativo: false, valor: 9.95 },
  { id: 3, nome: 'Luvas descartáveis', ativo: false, valor: 2.50 },
  { id: 3, nome: 'Papel toalha', ativo: true, valor: 2.50 },
];

function verificaProdutoAtivo (produto){
  return produto.ativo == true;
};

const produtosAtivos = produtos.filter(verificaProdutoAtivo);

console.log(produtosAtivos);
```

Executando a função **verificaProdutoAtivo** para cada elemento do array **produtos**.



```
const produtos = [
  { id: 1, nome: 'Açúcar', ativo: true, valor: 2.00 },
  { id: 2, nome: 'Álcool 70', ativo: false, valor: 9.95 },
  { id: 3, nome: 'Luvas descartáveis', ativo: false, valor: 2.50 },
  { id: 3, nome: 'Papel toalha', ativo: true, valor: 2.50 },
];

function verificaProdutoAtivo (produto){
  return produto.ativo == true;
};

const produtosAtivos = produtos.filter(verificaProdutoAtivo);

console.log(produtosAtivos);
```

Caso o retorno seja **true** a função **filter** vai guardar o produto no novo array.



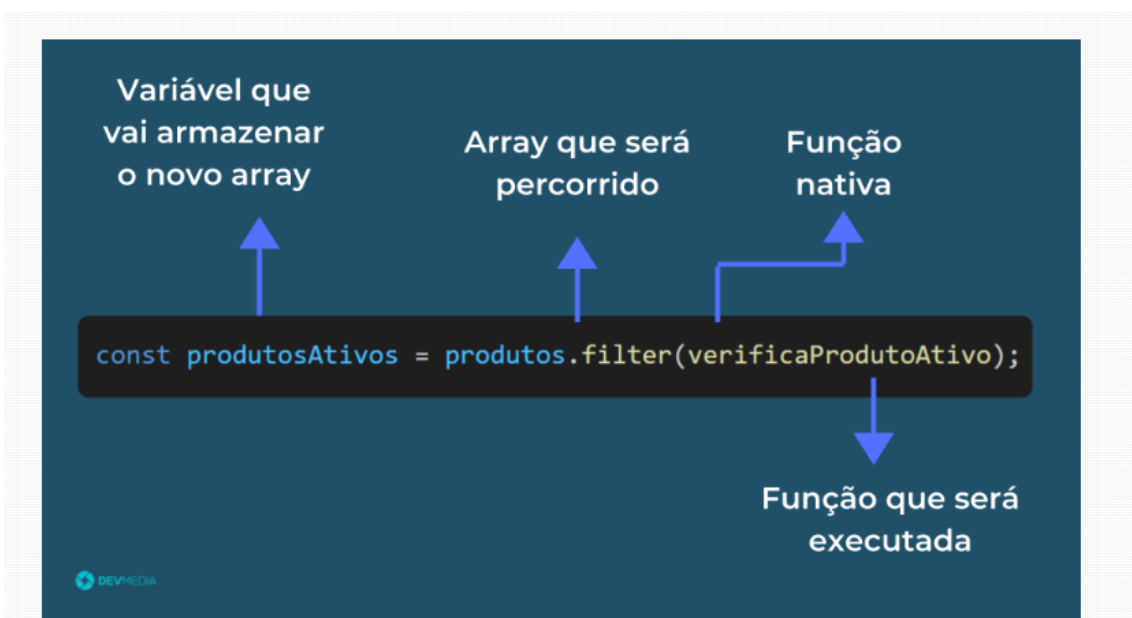
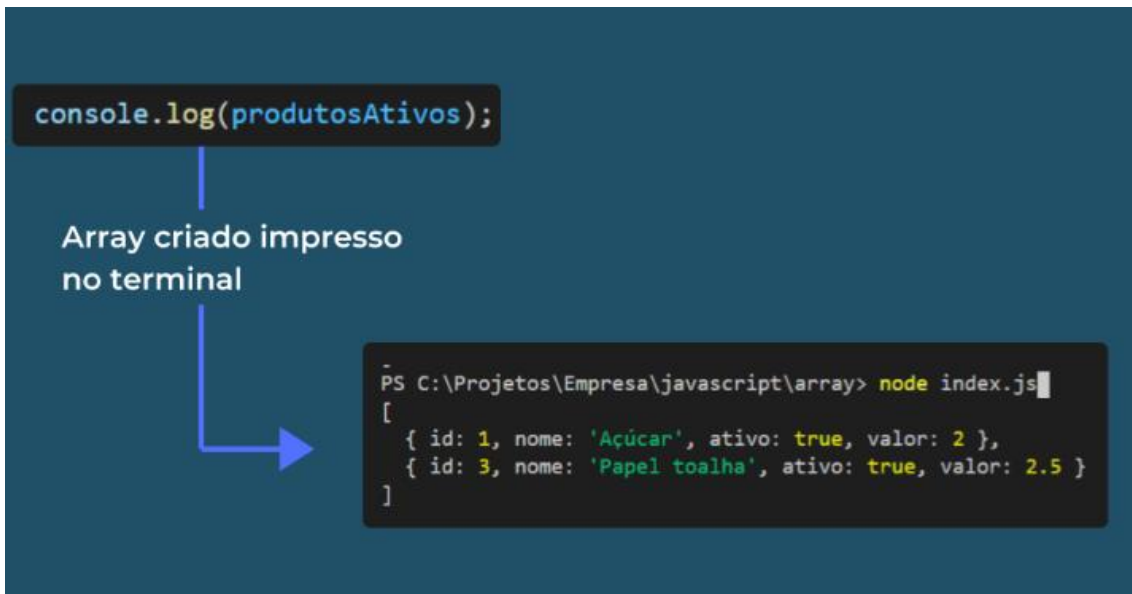


Figura 1. Sintaxe do filter

O elemento do array só será armazenado no novo array caso a função retorne true

Exemplo

Podemos utilizar `filter` para criar um array de carros da marca **Fiat**, conforme mostra a Figura 2.

```
const carros = [
  { marca: 'Fiat', modelo: 'Uno', anoFabricacao: 2015 },
  { marca: 'GM', modelo: 'Onix', anoFabricacao: 2018 },
  { marca: 'Ford', modelo: 'KA+', anoFabricacao: 2018 },
  { marca: 'Fiat', modelo: 'Cronos', anoFabricacao: 2020 },
];

function retornarCarroFiat(carro) {
  return (carro.marca == 'Fiat');
}

const carrosFiat = carros.filter( retornarCarroFiat );

console.log(carrosFiat);
```

Veja no **Código 2** o exemplo que apresentamos acima.

```
1  const carros = [  
2    { marca: 'Fiat', modelo: 'Uno', anoFabricacao: 2015 },  
3    { marca: 'GM', modelo: 'Onix', anoFabricacao: 2018 },  
4    { marca: 'Ford', modelo: 'KA+', anoFabricacao: 2018 },  
5    { marca: 'Fiat', modelo: 'Cronos', anoFabricacao: 2020 },  
6  ];  
7  
8  function retornarCarroFiat(carro) {  
9    return (carro.marca == 'Fiat');  
10 }  
11  
12 const carrosFiat = carros.filter( retornarCarroFiat );  
13  
14 console.log(carrosFiat);  
15  
16 /*  
17  * vai imprimir:  
18  [  
19    { marca: 'Fiat', modelo: 'Uno', anoFabricacao: 2015 },  
20    { marca: 'Fiat', modelo: 'Cronos', anoFabricacao: 2020 }  
21  ]  
22  */
```

Código 2. Filtrando os carros que são da Fiat

A função filter permite filtrar um array utilizando uma função para isso

Às vezes, você precisa utilizar algum procedimento específico nos elementos de um array para obter um novo array com os elementos modificados.

Em vez de iterar manualmente sobre o array usando um laço, você pode simplesmente usar o método `Array.map()` integrado.

O método `Array.map()` permite a você iterar sobre o array e modificar seus elementos usando uma função de callback. A função de callback será executada em cada um dos elementos do array.

Por exemplo, vamos supor que você tenha o array a seguir:

```
let arr = [3, 4, 5, 6];
```

Um array simples em JavaScript

Agora, imagine que você deseja multiplicar cada um dos elementos do array por 3. Você poderia usar um laço for, da seguinte maneira:

```
let arr = [3, 4, 5, 6];
```

```
for (let i = 0; i < arr.length; i++){  
  arr[i] = arr[i] * 3;  
}
```

```
console.log(arr); // [9, 12, 15, 18]  
Iterar sobre um array usando o laço for
```

No entanto, você pode usar o método `Array.map()` para obter o mesmo resultado. Vemos aqui um exemplo:

```
let arr = [3, 4, 5, 6];
```

```
let modifiedArr = arr.map(function(element){  
  return element * 3;  
});
```

```
console.log(modifiedArr); // [9, 12, 15, 18]  
Iterar sobre um array usando o método map()
```

O método `Array.map()` normalmente é usado para aplicar algumas alterações aos elementos, seja multiplicando-os por um número específico, como no código acima, seja fazendo outras operações que você necessite utilizar em sua aplicação.

[Como usar o método map\(\) em um array de objetos](#)

Por exemplo, você poderia ter um array de objetos que armazena os valores de `firstName` e `lastName` (nome e sobrenome) de seus amigos, conforme vemos abaixo:

```
let users = [  
  {firstName: "Susan", lastName: "Steward"},  
  {firstName: "Daniel", lastName: "Longbottom"},  
  {firstName: "Jacob", lastName: "Black"}  
];
```

Um array de objetos

Você pode usar o método `map()` para iterar sobre o array e unir os valores de `firstName` e `lastName` da seguinte maneira:

```
let users = [  
  {firstName: "Susan", lastName: "Steward"},  
  {firstName: "Daniel", lastName: "Longbottom"},  
  {firstName: "Jacob", lastName: "Black"}  
];
```

```
let userFullnames = users.map(function(element){  
  return `${element.firstName} ${element.lastName}`;  
})
```

```
console.log(userFullnames);
```

```
// ["Susan Steward", "Daniel Longbottom", "Jacob Black"]
```

Use o método map() para iterar sobre um array de objetos

O método map() passa mais do que apenas um elemento. Vejamos todos os argumentos passados pelo método map() para a função de callback.

A sintaxe completa do método map()

A sintaxe do método map() é a seguinte:

```
arr.map(function(elemento, índice, array){ }, this);
```

function() é a função de callback chamada para cada elemento do array. O método map() sempre passa o elemento atual, o índice do elemento atual e todo o objeto do array para ela.

O argumento this será usado dentro da função de callback. Por padrão, esse valor é undefined. Por exemplo, Aqui está a forma de mudar o valor de this para o número 80:

```
let arr = [2, 3, 5, 7]
```

```
arr.map(function(elemento, índice, array){  
  console.log(this) // 80  
}, 80);
```

Atribuindo um valor numérico ao argumento this do método map()

Você também pode testar outros argumentos usando console.log() se estiver interessado:

```
let arr = [2, 3, 5, 7]
```


```
arr.map(function(elemento, índice, array){  
  console.log(elemento);  
  console.log(índice);  
  console.log(array);  
  return elemento;  
}, 80);
```

Registrando os argumentos para ver os valores

Isso é tudo o que você precisa saber sobre o método Array.map(). Com frequência, você usará apenas o argumento elemento na função de callback, e ignorará o resto. É isso o que eu faço geralmente nos meus projetos diários :)

Suponha que você tenha um array de números e deseje criar um novo array contendo o quadrado de cada número. Vamos ver como isso pode ser feito com ambas as abordagens.

Exemplo com Arrow Function:

```
javascript  Copy code


const numeros = [1, 2, 3, 4, 5];

const quadrados = numeros.map(numero => numero * numero);

console.log(quadrados); // Saída: [1, 4, 9, 16, 25]
```

Neste exemplo, usamos uma arrow function no método `map` para criar um novo array `quadrados` que contém o quadrado de cada número. A arrow function é curta e concisa, tornando o código mais legível e fácil de entender.

Exemplo com Função Tradicional:

```
javascript  Copy code

const numeros = [1, 2, 3, 4, 5];

function calcularQuadrado(numero) {
  return numero * numero;
}

const quadrados = numeros.map(calcularQuadrado);

console.log(quadrados); // Saída: [1, 4, 9, 16, 25]
```

Neste caso, usamos uma função tradicional `calcularQuadrado` para calcular o quadrado de um número e, em seguida, usamos o método `map` para aplicar essa função a cada elemento do array. A abordagem funciona da mesma forma, mas a função tradicional é mais longa e adiciona complexidade ao código.

Explicação:

A vantagem de usar uma arrow function aqui está na simplicidade da sintaxe. Arrow functions são ideais para funções simples de uma única linha, como a que usamos para calcular o quadrado. Elas eliminam a necessidade de usar a palavra-chave `function` e permitem que você defina a função de forma mais concisa.

Portanto, ao escolher uma arrow function em vez de uma função tradicional, você obtém código mais limpo e legível, especialmente quando lida com tarefas simples e funções de retorno de chamada em que a sintaxe concisa é uma vantagem.