Métodos de arrays

Certified Developer
The Ultimate Tech Degree

DigitalHouse >
Coding School

Temas

1. map()
2. filter()
3. reduce()
4. forEach()

1 | map()

.map()

Este método recebe uma função como parâmetro (callback).

Faz um loop no array e retorna um novo array modificado.

As modificações serão aquelas que programamos em nossa função de callback.

```
{}
array.map(function(elemento){
    // definimos as modificações que queremos
    // aplicar sobre cada elemento do array
});
```

{código}

```
let numeros = [2, 4, 6];

let dobro = numeros.map(function(num){

    // Multiplicamos por 2 cada número
    return num * 2;

});

console.log(dobro); // [4,8,12]
```

Declaramos um array com três números.

{código}

```
let numeros = [2, 4, 6];

let dobro = numeros.map(function(num){

    // Multiplicamos por 2 cada número
    return num * 2;

});

console.log(dobro); // [4,8,12]
```

Aplicamos o método **.map()** no array **numeros**.

{código}

```
let numeros = [2, 4, 6];

let dobro = numeros.map(function(num){

    // Multiplicamos por 2 cada número
    return num * 2;

});

console.log(dobro); // [4,8,12]
```

Em seguida **passamos** uma função como **callback**.

Essa função, por sua vez, receberá um parâmetro que será os elementos do array.

No nosso caso será um número. Devido a isso, o nome do parâmetro é **num**

{código}

```
let numeros = [2, 4, 6];

let dobro = numeros.map(function(num){
    // Multiplicamos por 2 cada número
    return num * 2;
});

console.log(dobro); // [4,8,12]
```

Definimos o comportamento da função.

Ela irá executar 3 vezes (número de elementos do array) e retornará o valor multiplicado por 2.

{código}

```
let numeros = [2, 4, 6];

let dobro = numeros.map(function(num){

    // Multiplicamos por 2 cada número
    return num * 2;

});

console.log(dobro); // [4,8,12]
```

O valor será armazenado na variável **dobro**, onde terá os valores dobrados do array **numeros**.

{código}

```
let numeros = [2, 4, 6];

let dobro = numeros.map(function(num){

    // Multiplicamos por 2 cada número
    return num * 2;

});

console.log(dobro); // [4,8,12]
```

Exibimos o valor da variável, que se tornou um novo array, com os valores do array **numeros** multiplicados.

2 | filter()

.filter()

Este método também usa uma função como parâmetro.
Realiza um loop no array e filtra os elementos com base em uma condição existente na callback.

Retorna um novo array que contém apenas os elementos que atendem a essa condição, ou seja, poderá ter um número menor de elementos comparado ao anterior.

```
{}
array.filter(function(elemento){
    // definimos a condição que queremos aplicar
    // como filtro para a criação do novo array
});
```

{código}

```
let idades = [22, 8, 17, 14, 30];

let maiores = idades.filter(function(idade){
    return idade > 18;
});

console.log(maiores); // [22, 30]
```

Declaramos um array **idades** com diversos elementos.

{código}

```
let idades = [22, 8, 17, 14, 30];

let maiores = idades.filter(function(idade){
    return idade > 18;
});

console.log(maiores); // [22, 30]
```

Aplicamos o método **.filter()** no array de idades.

{código}

```
let idades = [22, 8, 17, 14, 30];

let maiores = idades.filter(function(idade){
    return idade > 18;
});

console.log(maiores); // [22, 30]
```

Em seguida, **passamos** uma função como **callback**.

Essa função, por sua vez, receberá um parâmetro que será os elementos do array.

No nosso caso será uma idade. Devido a isso, o nome do parâmetro é **idade.**

{código}

```
let idades = [22, 8, 17, 14, 30];

let maiores = idades.filter(function(idade){
    return idade > 18;
});

console.log(maiores); // [22, 30]
```

Definimos a condição interna da função.

A função será executada 5 vezes (número de elementos do array), e irá filtrar segundo a condição de idade ser maior que 18.

Os que não atenderem à condição serão excluídos.

{código}

```
let idades = [22, 8, 17, 14, 30];

let maiores = idades.filter(function(idade){
    return idade > 18;
});

console.log(maiores); // [22, 30]
```

Os valores filtrados serão armazenados na variável **maiores.**

{código}

```
let idades = [22, 8, 17, 14, 30];

let maiores = idades.filter(function(idade){

    return idade > 18;

});

console.log(maiores); // [22, 30]
```

Mostramos apenas os valores filtrados contidos no array **maiores**.

3 | reduce()

.reduce()

Este método percorre a array e retorna um único valor.
Ele recebe uma callback para ser executada em cada elemento do array.

Este, por sua vez, recebe dois parâmetros: um acumulador e o elemento atual que está percorrendo.

```
{}
array.reduce(function(acumulador, elemento){
    // definimos o comportamento que queremos
    // implementar sobre o acumulador e o elemento
});
```

{código}

```
let numeros = [5, 7, 16];

let soma = numeros.reduce(

  function(acumulador, numero){
    return acumulador + numero;
  }
);

console.log(soma); // 28
```

Declaramos um array **numeros** com diversos elementos.

{código}

```
let numeros = [5, 7, 16];

let soma = numeros.reduce(
  function(acumulador, numero){
    return acumulador + numero;
  }
);

console.log(soma); // 28
```

Aplicamos o método **.reduce()** ao array **numeros**.

{código}

```
let numeros = [5, 7, 16];

let soma = numeros.reduce(
  function(acumulador, numero){
    return acumulador + numero;
  }
);

console.log(soma); // 28
```

Em seguida **passamos** uma função como **callback**.

Essa função, por sua vez, receberá dois parâmetros.

O **acumulador** é o ultimo item retornado, e o **numero,** o elemento atual do array.

{código}

```
let numeros = [5, 7, 16];

let soma = numeros.reduce(

  function(acumulador, numero){
    return acumulador + numero;
  }
);

console.log(soma); // 28
```

Retornamos o **acumulador** (retorno do elemento anterior), somado com **numero** (elemento atual).

{código}

```
let numeros = [5, 7, 16];

let soma = numeros.reduce(
  function(acumulador, numero){
    return acumulador + numero;
  }
);

console.log(soma); // 28
```

Atribuímos o resultado **total** na variável **soma**.

{código}

```
let numeros = [5, 7, 16];

let soma = numeros.reduce(

  function(acumulador, numero){
    return acumulador + numero;
  }
);

console.log(soma); // 28
```

Por fim, exibimos o valor da variável **soma**.

4 | forEach()

.forEach()

O objetivo deste método é iterar em um array.

Ele recebe uma callback como parâmetro e, ao contrário dos métodos anteriores, não retorna nada.

```
{}
array.forEach(function(elemento){
    // definimos o comportamento que queremos
    // implementar sobre cada elemento
});
```

{código}

```
var paises = ['Brasil', 'Cuba', 'Peru'];

paises.forEach(
  function(pais){

    console.log(pais);
});
```

Declaramos um array **paises** com alguns elementos.

{código}

```
var paises = ['Brasil', 'Cuba', 'Peru'];

paises.forEach(
  function(pais){

    console.log(pais);
});
```

Aplicamos o método **.forEach()** no array países.

{código}

```
var paises = ['Brasil', 'Cuba', 'Peru'];

paises.forEach(
  function(pais){

    console.log(pais);
});
```

Passamos uma função para o método **.forEach()** como um parâmetro (**callback**).

Esta função recebe um parâmetro, que representará cada elemento do array, neste caso, cada país.

{código}

```
var paises = ['Brasil', 'Cuba', 'Peru'];

paises.forEach(
  function(pais){

    console.log(pais);
});
```

Definimos o comportamento interno da função.

Nesse caso, colocamos para exibir o país.

{código}

```
var paises = ['Brasil', 'Cuba', 'Peru'];
```

```
paises.forEach(
  function(pais){

   console.log(pais);
});
```

O método **.forEach()** irá exibir todos os elementos do array.

```
Brasil
Cuba
Peru
```


DigitalHouse>
Coding School