

# MÉTODOS DE ARRAYS II

“

JavaScript nos provee de varios **métodos** para ejecutar sobre arrays, dándonos así un abanico de herramientas para poder trabajar con ellos.



# ¡SPOILER ALERT!

Antes de conocer esos métodos, es necesario espiar un poco la definición de **callback**.

Como ya sabemos, las funciones pueden recibir uno o más parámetros. En caso de recibirlos, éstos pueden ser, entre otros: un número, un string, un booleano o también... **¡una función!**

Cuando una método o función recibe a una **función como parámetro**, a esa función se la conoce como **callback**.



1.

`.map()`

## .map()

Este método recibe una función como parámetro (*callback*).

Recorre el array y **devuelve** un **nuevo array** modificado.

Las modificaciones serán aquellas que programemos en nuestra función de callback.

```
array.map(function(elemento){  
    // definimos las modificaciones que queremos  
    // aplicar sobre cada elemento del array  
})
```

{ }

```
{ código }
```

```
var numeros = [2, 4, 6];  
var dobleNumeros = numeros.map(function(num){  
    // Multiplicamos por 2 cada número  
    return num * 2;  
});  
  
console.log(dobleNumeros); // [4,8,12]
```

```
{ código }
```

```
var numeros = [2,4,6];  
var dobleNumeros = numeros.map(function(num){  
    return num * 2;  
});  
  
console.log(dobleNumeros); // [4, 8, 12]
```

Declaramos la variable numeros y almacenamos un array con tres números.

```
{ código }
```

```
var numeros = [2,4,6];  
var dobleNumeros = numeros.map(function(num){  
    return num * 2;  
});  
  
console.log(dobleNumeros); // [4, 8, 12]
```

Aplicamos el método **map** al array de números.



```
{ código }
```

```
var numeros = [2,4,6];  
var dobleNumeros = numeros.map(function(num){  
    return num * 2;  
});  
  
console.log(dobleNumeros); // [4, 8, 12]
```

Al **map()** le pasamos una función como parámetro (callback).

Esa función, a su vez, recibe un parámetro (puede tener el nombre que queramos).

El parámetro va a representar a cada elemento de nuestro array, en este caso, un número.

```
{ código }
```

```
var numeros = [2,4,6];  
var dobleNumeros = numeros.map(function(num){  
    return num * 2;  
});  
  
console.log(dobleNumeros); // [4, 8, 12]
```

Definimos el comportamiento interno que va a tener la función.

La función se va a ejecutar 3 veces: una por cada elemento de este array, y a cada uno lo va a multiplicar por 2.

```
{ código }
```

```
var numeros = [2,4,6];  
var dobleNumeros = numeros.map(function(num){  
    return num * 2;  
});  
  
console.log(dobleNumeros); // [4, 8, 12]
```

En la variable **dobleNumeros** vamos a almacenar el array que nos va a devolver el método **map**.

```
{ código }
```

```
var numeros = [2,4,6];  
var dobleNumeros = numeros.map(function(num){  
    return num * 2;  
});
```

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

Mostramos por consola la variable **dobleNumeros** que almacena un nuevo array con la misma cantidad de elementos que el original. pero con valores modificados.

A thick, solid green diagonal stripe runs from the top-left towards the bottom-right, occupying the right half of the slide.

# 2.

## `.filter()`

## .filter()

Este método también recibe una función como parámetro.

Recorre el array y **filtra** los elementos según una condición que exista en el callback.

**Devuelve** un **nuevo array** que contiene únicamente los elementos que hayan cumplido con esa condición. Es decir que nuestro nuevo array puede contener menos elementos que el original.

```
array.filter(function(elemento){  
    // definimos la condición que queremos utilizar  
    // como filtro para cada elemento del array  
});
```

{ }

```
{ código }
```

```
var edades = [22, 8, 17, 14, 30];  
var mayores = edades.filter(function(edad){  
    return edad > 18;  
});  
  
console.log(mayores); // [22, 30]
```

```
{ código }
```

```
var edades = [22, 8, 17, 14, 30];  
var mayores = edades.filter(function(edad){  
    return edad > 18;  
});  
  
console.log(mayores); // [22, 30]
```

Declaramos la variable edades y almacenamos un array con cinco números.



```
{ código }
```

```
var edades = [22, 8, 17, 14, 30];  
var mayores = edades.filter(function(edad){  
    return edad > 18;  
});  
  
console.log(mayores); // [22, 30]
```

Aplicamos el método **filter** al array de edades.

```
{ código }
```

```
var edades = [22, 8, 17, 14, 30];  
var mayores = edades.filter(function(edad){  
    return edad > 18;  
});  
  
console.log(mayores); // [22, 30]
```

Al método **filter()** le pasamos una función como parámetro (callback).

Esa función, a su vez, recibe un parámetro (puede tener el nombre que queramos).

El mismo va a representar a cada elemento de nuestro array, en este caso, una edad.

```
{ código }
```

```
var edades = [22, 8, 17, 14, 30];  
var mayores = edades.filter(function(edad){  
    return edad > 18;  
});  
  
console.log(mayores); // [22, 30]
```

Definimos el comportamiento interno que va a tener esa función.

La función se va a ejecutar 5 veces: una por cada elemento de este array, y los va a filtrar según la condición que definimos: que las edades se mayores a 18.

Esto quiere decir que las que no cumplan con la condición ( $\text{edad} > 18 \Rightarrow \text{false}$ ), serán excluidas.

```
{ código }
```

```
var edades = [22, 8, 17, 14, 30];  
var mayores = edades.filter(function(edad){  
    return edad > 18;  
});  
  
console.log(mayores); // [22, 30]
```

En la variable **mayores** almaceno el array nuevo que me devuelve el método filter.

```
{ código }
```

```
var edades = [22, 8, 17, 14, 30];  
var mayores = edades.filter(function(edad){  
    return edad > 18;  
});
```

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

Muestro por consola la variable **mayores** que tiene el array con los elementos que cumplieron con la condición establecida.

“

Es una **buena práctica** utilizar **nombres que tengan sentido** para nuestras **variables**.



```
var mayores = edades.filter(function(edad){  
    return edad > 18;  
});
```



De esa manera **queda más claro** para qué se van a utilizar.



3.

`.reduce()`

## .reduce()

Este método recorre el array y devuelve un **único valor**.

Recibe un callback que se va a ejecutar sobre cada elemento del array. El mismo, a su vez, recibe dos parámetros: un **acumulador** y el **elemento actual** que esté recorriendo.

```
array.reduce(function(acumulador, elemento){  
    // definimos el comportamiento que queremos  
    // implementar sobre el acumulador y el elemento  
});
```

{ }



```
{ código }
```

```
var numeros = [5, 7, 16];  
var suma = numeros.reduce(function(acum, num){  
    return acum + num;  
});  
  
console.log(suma); // 28
```

```
{ código }
```

```
var numeros = [5, 7, 16];  
var suma = numeros.reduce(function(acum, numero){  
    return acum + numero;  
});  
  
console.log(suma); // 28
```

Declaramos la variable `numeros` y le asignamos un array con tres elementos.

```
{ código }
```

```
var numeros = [5, 7, 16];  
var suma = numeros.reduce(function(acum, numero){  
    return acum + numero;  
});  
  
console.log(suma); // 28
```

Le aplicamos el método **reduce()** al array de numeros.

```
{ código }
```

```
var numeros = [5, 7, 16];  
var suma = numeros.reduce(function(acum, numero){  
    return acum + numero;  
});  
  
console.log(suma); // 28
```

Al **reduce()** le pasamos una función como parámetro (callback).

Esa función recibe dos parámetros (pueden tener el nombre que queramos).

El primero va a representar el acumulador, el segundo el elemento que esté recorriendo en ese momento, en este caso un número.

```
{ código }
```

```
var numeros = [5, 7, 16];  
var suma = numeros.reduce(function(acum, num){  
    return acum + numero;  
});  
  
console.log(suma); // 28
```

Definimos el comportamiento interno de la función. En este caso, queremos devolver la **suma** total de los elementos.

El acumulador irá almacenando el resultado y por cada iteración sumará el elemento actual.

```
{ código }
```

```
var numeros = [5, 7, 16];  
var suma = numeros.reduce(function(acum, num){  
    return acum + numero;  
});  
  
console.log(suma); // 28
```

En la variable **suma** almacenamos lo que devuelva el método **reduce()** al aplicarlo al array **numeros**.

```
{ código }
```

```
var numeros = [5, 7, 16];  
var suma = numeros.reduce(function(acum, num){  
    return acum + numero;  
});
```

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

Mostramos por consola la variable **suma** que tiene la suma total de los números del array.



4.

**.forEach()**



## .forEach()

La finalidad de este método es iterar sobre un array.

Recibe un callback como parámetro y, a diferencia de los métodos anteriores, éste **no retorna nada**.

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

{ }

```
{ código }
```

```
var paises = ['Argentina', 'Brasil', 'Colombia'];  
paises.forEach(function(pais){  
    console.log(pais);  
});
```

```
{ código }
```

```
var paises = ['Argentina', 'Brasil', 'Colombia'];  
paises.forEach(function(pais){  
    console.log(pais);  
});
```

Declaro la variable `paises` y le asigno un array con tres elementos.

```
{ código }
```

```
var paises = ['Argentina','Brasil','Colombia'];  
paises.forEach(function(pais){  
    console.log(pais);  
});
```

Le aplico el método **forEach()** al array de paises.

```
{ código }
```

```
var paises = ['Argentina','Brasil','Colombia'];  
paises.forEach(function(pais){  
    console.log(pais);  
});
```

Al método **forEach()** le pasamos una función como parámetro (callback).

Esa función recibe un parámetro, que va a representar a cada elemento del array, en este caso, a cada país.

```
{ código }
```

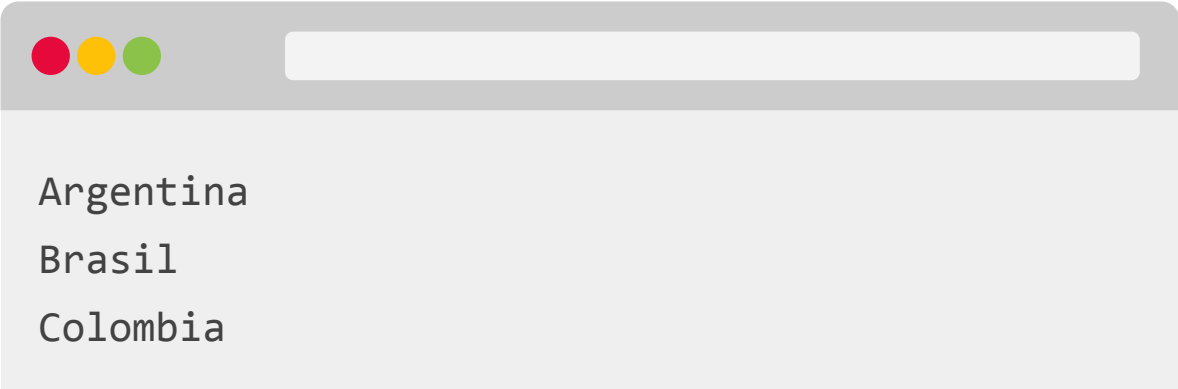
```
var paises = ['Argentina','Brasil','Colombia'];  
paises.forEach(function(pais){  
    console.log(pais);  
});
```

Defino el comportamiento interno de la función.  
En este caso, quiero mostrar por consola a cada país.

```
{ código }
```

```
var paises = ['Argentina', 'Brasil', 'Colombia'];  
paises.forEach(function(pais){  
    console.log(pais);  
});
```

El método **forEach()** en este caso imprimirá por consola todos los elementos del array.



```
Argentina  
Brasil  
Colombia
```