Clase 16: Métodos y Propiedades

Índice

Metodos Array Avanzado:

- Map
- Filter
- Find
- Reduce
- foreach

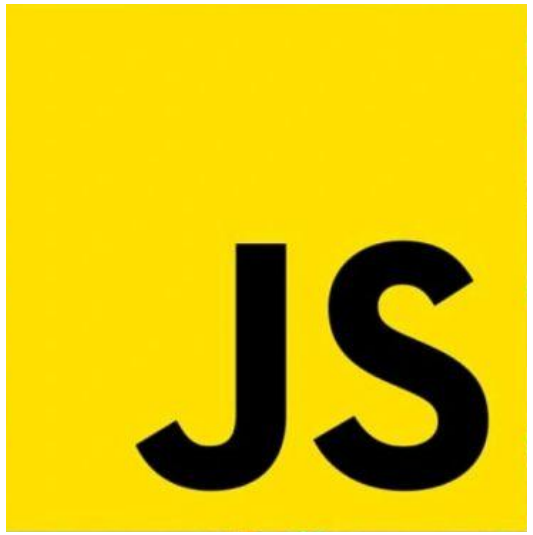
Call Back:

- Concepto

Métodos de Arrays (Avanzados)

Métodos de Arrays

- JavaScript nos provee de varios métodos para ejecutar sobre arrays, dándonos así un abanico de herramientas para poder trabajar con ellos.
- Si bien la realidad es que la cantidad de métodos disponibles es muy grande y variada, nosotros nos enfocaremos en los que consideramos son los más importantes y utilizados en la actualidad. Estos son:
  - ✔ map
  - ✔ filter
  - ✔ reduce
  - ✔ forEach



Métodos de Arrays

Importante:

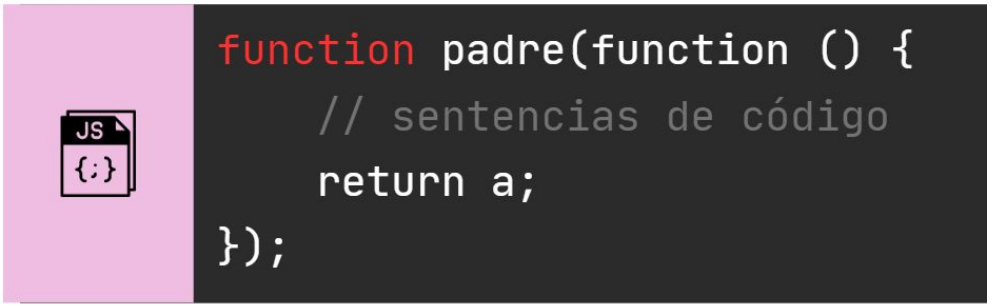
Para poder utilizar estos métodos (map, filter, reduce y forEach) es necesario antes de aprenderlos, tener bien presente la definición de **callback**.
Pero… que significa este termino?

Métodos de Arrays

Callback:

Como ya sabemos, las funciones pueden recibir uno o más parámetros.
En caso de recibirlos, estos pueden ser (entre otros): un número, un string, un booleano o también... ¡una función!
Cuando un método o función recibe a una función como parámetro, a esa función se la conoce como callback.
Y es la función que recibe dicho Callback, quien se encarga de ejecutar la función pasada cuando esto sea necesario.

Un Callback generalmente se ve asi:



Como pueden observar, función padre esta recibiendo como parámetro una función que no tiene nombre (función anónima)

Método .map()

Métodos de Arrays

.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
});
```

Métodos de Arrays

¿Qué sucede dentro de un callback?

Esa lógica la implementas vos en función de lo que estes queriendo hacer.
Por ejemplo, si quisieras multiplicar por dos cada uno de los números de este ejemplo dentro del callback, vas a programar este comportamiento.
Y como resultado obtendrás un array completamente nuevo con esa lógica implementada

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

let dobleDeLosNumeros = numeros.map(function (unNumero) {
    return unNumero * 2;
});

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

```

Métodos de Arrays

Ahora explicamos paso a paso el siguiente código

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

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

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

Métodos de Arrays

{Codigo}

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

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

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

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

Métodos de Arrays

{Codigo}

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

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

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

Aplicamos el método map() al array de números.

Métodos de Arrays

{Codigo}

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

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

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

Al método map() pasamos una función como parámetro (callback).
Esa función, a su vez, recibe un parámetro –puede tener el nombre que queramos –.
Este va a representar a cada elemento de nuestro array, en este caso, un número.

Métodos de Arrays

{Codigo}

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

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

console.log(numerosMultiplicados); // [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.

Métodos de Arrays

{Codigo}

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

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

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

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

Métodos de Arrays

{Codigo}

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

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

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

Mostramos por consola la variable numerosMultiplicados que contiene un nuevo array con la misma cantidad de elementos que el original, pero con valores modificados.

Método .filter()

Métodos de Arrays

.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
});
```

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
const edades = [22, 8, 17, 14, 30];

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

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

Métodos de Arrays

{Codigo}

```
const edades = [22, 8, 17, 14, 30];

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

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

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

Métodos de Arrays

{Codigo}

```
const edades = [22, 8, 17, 14, 30];

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

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

Aplicamos el método filter al array de edades.

Métodos de Arrays

{Codigo}

```
const edades = [22, 8, 17, 14, 30];

const 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–.

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

Métodos de Arrays

{Codigo}

```
const edades = [22, 8, 17, 14, 30];

const 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 sean mayores a 18.

Esto quiere decir que las que no cumplan con la condición, serán excluidas.

Métodos de Arrays

{Codigo}

```
const edades = [22, 8, 17, 14, 30];

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

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

En la variable mayores almacenamos el array que nos devuelve el método filter.

Métodos de Arrays

{Codigo}

```
const edades = [22, 8, 17, 14, 30];

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

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

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

Método .reduce()

Métodos de Arrays

.reduce()

Este método recorre el array y devuelve un único valor.
Recibe un callback que se va a ejecutar sobre cada elemento del array.
Este, a su vez, recibe cuatro argumentos: un acumulador, elemento actual, índice actual y array que esté recorriendo.


```
array.reduce(function(acumulador, elemento, índice, array)
{
    // definimos el comportamiento que queremos
    // implementar sobre el acumulador y el elemento
 // podemos indicar el valor inicial que por defecto es 0
}, 0);
```

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
const nums = [5, 7, 16];

const suma = nums.reduce(function(acum, num){
    return acum + num;
});

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

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
const nums = [5, 7, 16];

const suma = nums.reduce(function(acum, num){
    return acum + num;
});

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

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

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
const nums = [5, 7, 16];

const suma = nums.reduce(function(acum, num){
    return acum + num;
});

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

Le aplicamos el método reduce() al array de números.

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
const nums = [5, 7, 16];

const suma = nums.reduce(function(acum, num){
   return acum + num;
});

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.

Métodos de Arrays

Veamos el desarrollo paso a paso:

Ejemplo:
Acumulador + unNumero → literación 1
0 + 5
Acumulador + unNumero → literación 2
5 + 7
Acumulador + unNumero → literación 3
12 + 16

```
const nums = [5, 7, 16];

const suma = nums.reduce(function(acum, num){
  return acum + num;
});

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.

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
const nums = [5, 7, 16];

const suma = nums.reduce(function(acum, num){
    return acum + num;
});

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

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

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
const nums = [5, 7, 16];

const suma = nums.reduce(function(acum, num){
    return acum + num;
});

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

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

Método .forEach()

Métodos de Arrays

.forEach()

La finalidad de este método es iterar sobre un array.
Recibe un callback como parámetro y, a diferencia de los métodos anteriores, este no retorna nada.


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

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
var paises = ['Argentina', 'Cuba', 'Perú'];

paises.forEach(function(pais){
   console.log(pais);
});
```

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
var paises = ['Argentina', 'Cuba', 'Perú'];

paises.forEach(function(pais){
  console.log(pais);
});
```

Declaramos la variable paises y almacenamos un array con tres países.

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
var paises = ['Argentina', 'Cuba', 'Perú'];

paises.forEach(function(pais){
   console.log(pais);
});
```

Le aplicamos el método forEach() al array de países.

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
var paises = ['Argentina', 'Cuba', 'Perú'];

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.

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
var paises = ['Argentina', 'Cuba', 'Perú'];

paises.forEach(function(pais){
  console.log(pais);
});
```

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

Métodos de Arrays

Veamos el desarrollo paso a paso:

```
var paises = ['Argentina', 'Cuba', 'Perú'];
```

```
paises.forEach(function(pais){
  console.log(pais);
});
```

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

Terminal

```
Argentina
Cuba
Peru
```

Método .find()

Métodos de Arrays

Método .find() :

Este método toma una función de callback y llama a esa función para cada elemento que recorre dentro del arreglo al que está vinculado.
Cuando **encuentra una coincidencia** (en otras palabras, la función callback devuelve true), el método **devuelve ese elemento particular del arreglo** e inmediatamente **rompe el bucle**.
Así que el método find() devuelve el primer elemento dentro de un arreglo que satisface la función callback.

Métodos de Arrays

Método .find() :

La función callback puede tomar los siguientes parámetros:

- currentItem: Es el elemento del arreglo sobre el que se está iterando actualmente.
- index: Esta es la posición de índice de currentItem dentro del arreglo.
- array: Representa el arreglo de destino junto con todos sus elementos.

Métodos de Arrays

Cómo utilizar el método find():

Ejemplo: Cómo obtener un solo elemento con find():

Supongamos que tienes un perro que se ha perdido.
Lo denuncias a las autoridades competentes y estas reúnen a un grupo de perros recuperados.
Para poder encontrar a tu perro, necesitas proporcionar información única sobre él. Por ejemplo, la raza de su perro (un chihuahua) puede servir para identificarlo.
Podemos expresar este escenario en JavaScript utilizando una colección de arreglos.
El arreglo llamado **perrosEncontrados** contendrá todos los nombres de los perros recuperados, así como sus respectivas razas. Y utilizaremos el método find() para encontrar el perro que es un chihuahua desde dentro del arreglo.

Métodos de Arrays

Cómo utilizar el método find():

Ejemplo: Cómo obtener un solo elemento con find():

```
let perrosEncontrados = [{
    breed: "Beagle",
    color: "blanco"
  },

  {
    raza: "Chihuahua",
    color: "amarillo"
  },

  {
    breed: "Pug",
    color: "negro"
  },
]
```

```
function encuentraMiPerro(perro) {
  return perro.raza === "Chihuahua"
}

let miPerro = perrosEncontrados.find(perro => encuentraMiPerro(perro));

console.log(miPerro);


/*

{
  raza: "Chihuahua",
  color: "amarillo"
}
```

Hay algo muy importante que hay que recordar sobre find():
deja de ejecutarse una vez que la función callback devuelve una declaración como true. Es decir, El método find() deja de iterar cuando se encuentra una coincidencia.

Métodos de Arrays

Cómo utilizar el método find():

Ejemplo: Cómo encontrar un elemento por su índice:

En este ejemplo, encontraremos y devolveremos el punto perteneciente a 'David' desde dentro del arreglo utilizando su valor de índice único.

Esto demuestra una de las formas en que podemos utilizar la propiedad index dentro de nuestra función callback con el método find():

Métodos de Arrays

Cómo utilizar el método find():

Ejemplo: Cómo obtener un solo elemento con find():

```
let posicionesReservadas = [{
    nombre: "Ana",
    edad: 24
  },

  {
    nombre: "Beth",
    edad: 22
  },

  {
    nombre: "Cara",
    edad: 25
  },

  {
    nombre: "David",
    edad: 30
  },

  {
    nombre: "Ethan",
    edad: 26
  }
]
```

```
function encontrarPorIndice(persona, indice) {
  return indice === 3
}



let miPosicion = posicionesReservadas.find((persona, indice) => encontrarPorIndice(persona,

console.log(miPosicion);

/*
{
  edad: 30,
  nombre: "David"
}
*/
```

Veamos mas sobre los Objetos en VSC!

Momento de poner a prueba lo aprendido!