

Map JS

El método `map()` de JavaScript es una función de la clase `Array` que permite aplicar una función a cada elemento de un array y devolver un nuevo array con los resultados. Es muy útil para transformar un array de una forma rápida y sencilla.

Aquí tienes un ejemplo de cómo se usa `map()`:

```
const numbers = [1, 2, 3, 4, 5];

const doubledNumbers = numbers.map(number => number * 2);

console.log(doubledNumbers); // [2, 4, 6, 8, 10]
```

En este ejemplo, hemos creado un array `numbers` con cinco números. Luego, hemos utilizado `map()` para aplicar la función que duplica cada número a cada elemento del array y guardar el resultado en un nuevo array `doubledNumbers`. Al final, hemos imprimido este nuevo array en la consola y podemos ver que cada número del array original ha sido duplicado.

Otro ejemplo de cómo se puede usar `map()` es para transformar un array de objetos:

```
const users = [
  { name: 'John', age: 30 },
  { name: 'Jane', age: 25 },
  { name: 'Bob', age: 35 }
];

const names = users.map(user => user.name);

console.log(names); // ['John', 'Jane', 'Bob']
```

En este caso, tenemos un array `users` con objetos que tienen dos propiedades: `name` y `age`. Usamos `map()` para extraer el valor de la propiedad `name` de cada objeto y guardarlo en un nuevo array `names`. Al final, imprimimos este nuevo array en la consola y podemos ver que sólo contiene los nombres de los usuarios.

Avanzando en el uso de map

Os explicaremos cómo funciona con un ejemplo simple. Supongamos que ha recibido un **array** que **contiene varios objetos**, cada uno de los cuales representa a una persona.

```
// Lo que tenemos

var officers = [
  { id: 20, name: 'Captain Piatt' },
  { id: 24, name: 'General Veers' },
  { id: 56, name: 'Admiral Ozzel' },
  { id: 88, name: 'Commander Jerjerrod' }
];

// Lo que necesitamos [20, 24, 56, 88]
```

Hay múltiples formas de lograr esto. Es posible hacerlo creando un array vacío y luego usando `.forEach()`, `.for(... of)` o un simple `.for()` para cumplir su objetivo.

Usando **`.forEach()`**:

```
var officersIds = [];

officers.forEach(officer => {
  officersIds.push(officer.id);
});
```

Usando **.map()**:


```
const officersIds = officers.map(officer => officer.id);
```

Entonces, ¿cómo funciona **.map()**? Básicamente **recibe dos argumentos**, una devolución de llamada y un contexto opcional (se considerará así en la devolución de llamada) que no utilicé en el ejemplo anterior. La devolución de llamada se ejecuta para cada valor en el **array** y devuelve cada nuevo valor en el **array** resultante.

```
var arr = [{
  id: 1,
  name: 'bill'
}, {
  id: 2,
  name: 'ted'
}]

var result = arr.map(person => ({ value: person.id, text: person.name }));

console.log(result)
```



```
▼ (2) [{...}, {...}] ⓘ
  ► 0: {value: 1, text: "bill"}
  ► 1: {value: 2, text: "ted"}
    length: 2
  ► __proto__: Array(0)
```

Tenemos que tener en cuenta que el **array** que nos devuelve siempre tendrá la misma longitud que el original. Pero es un **array** nuevo, el original no se ha modificado.