



# Javascript S5 | Arrays en ES6

Después de esta lección podrás:

1. Manejo de Arrays.
2. Conocer los métodos más importantes de un Array.
3. Uso de las novedades de ES6 Map - Filter - Reduce.
4. Combinar las novedades de ES6.

## Métodos de Array

Un **array** es un tipo de **variable** que nos permite **agrupar** un conjunto de **variables**. Los arrays se definen indicando el conjunto de variables que queremos almacenar entre corchetes. Por ejemplo, vamos a definir un array con un cartel de festival:

```
let bbkLive = [ "Belako", "Vetusta", "Pulp", "Greta Van Fleet" ];
```

Para **acceder** a un **elemento concreto del array** se puede hacer mediante el índice de la posición que ocupa indicado entre corchetes ***nombreDelArray[posición]***, empezando a contar por el 0.

```
let bbkLive = [ "Belako", "Vetusta", "Pulp", "Greta Van Fleet" ];
console.log("El primer grupo es: " + bbkLive[0]);
console.log("El tercer grupo es: " + bbkLive[2]);
```

## Arrays: recorrido con for

Una **propiedad** muy útil de los **arrays** es ***"length"***. Mediante length podemos **conocer la longitud** del array, lo cual combinado con un bucle "for" nos permite recorrer todas sus posiciones:

```
let bbkLive = [ "Belako", "Vetusta", "Pulp", "Greta Van Fleet" ];

for (var indice = 0; indice < bbkLive.length; indice++) {
    console.log("El grupo número " + indice + " es " + bbkLive[indice]);
}
```

Esto nos imprimirá cada uno de los elementos del array.

## Arrays: añadiendo elementos

En el ejemplo anterior hemos definido directamente el array con todo su contenido, pero en muchas ocasiones esto no se adaptará a nuestras necesidades. Son muchos los casos en los que debemos **añadir elementos a los arrays de forma dinámica**.

Para ello disponemos de la función “**push**” y la función “**unshift**”. Vamos a ver cómo crear un array vacío y posteriormente añadirle valores. **Push lo añade al final, unshift al principio.**

```
var bbkLive = [];  
  
bbkLive.push("León Benavente");  
bbkLive.push("Rusowsky");  
  
console.log(bbkLive);  
  
bbkLive.unshift("Rusowsky");  
  
console.log(bbkLive);
```

## Arrays: eliminando elementos

En caso de que necesitemos **eliminar** un **elemento de un array**, disponemos de las funciones contrarias:

```
var fib = ["Franz Ferdinand", "Arctic Monkeys", "Love of lesbian"];  
var ultimo = fib.pop();  
var primero = fib.shift();  
console.log(fib);
```

Con la llegada de ES6 no estamos ligados al uso de los bucles for, sino que tenemos alternativas muy interesantes que al menos debemos conocer.

## Usando → .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 Piett' },
  { 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(function (officer) {
  officersIds.push(officer.id);
});
```

Usando **`.map()`**:

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

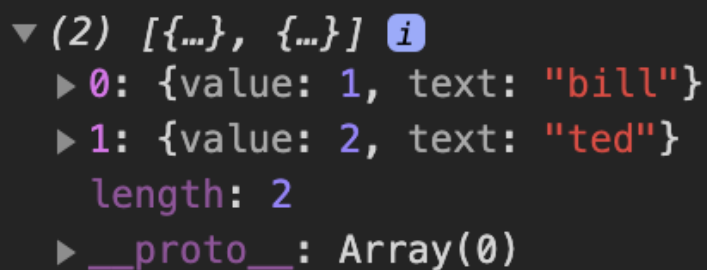
Entonces, ¿cómo funciona **`.map()`**? Básicamente **recibe dos argumentos**, una devolución de llamada y un contexto opcional (se considerará así en la devolución de llamada) que no utilicé en el ejemplo anterior. La devolución de

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.

## Usando → **.reduce()**

Al igual que **.map ()**, **.reduce ()** también ejecuta una devolución de llamada para cada elemento de un **array**. Lo diferente aquí es que **reduce el resultado** de esta devolución de llamada (el acumulador) de un elemento del **array** a otro.

El acumulador puede ser prácticamente cualquier cosa

(entero, cadena, objeto, etc.) y debe instanciarse o pasarse al llamar a `.reduce()`.

¡Hora de un ejemplo! Digamos que tienes una **Array** con estos pilotos y sus respectivos años de experiencia:

```
var pilots = [
  { id: 10, name: "Poe Dameron", years: 14, },
  { id: 2, name: "Temmin 'Snap' Wexley", years: 30, },
  { id: 41, name: "Tallissan Lintra", years: 16, },
  { id: 99, name: "Ello Asty", years: 22, }
];
```

Necesitamos conocer el **total de años de experiencia** de todos ellos. Con **`.reduce()`**, es bastante sencillo:

```
var totalYears = pilots.reduce(function (accumulator, pilot) {
  return accumulator + pilot.years;}, 0);
```

Tenemos en cuenta que hemos establecido el **valor inicial en 0**. También podría haber usado una variable existente si fuera necesario. Después de ejecutar la devolución de llamada para cada elemento del array, `reduce` devolverá el valor final de nuestro acumulador (en nuestro caso: 82).

Veamos cómo se puede acortar esto con las funciones de flecha de ES6:

```
const totalYears = pilots.reduce((acc, pilot) => acc + pilot.years, 0);
```

Ahora digamos que quiero encontrar qué piloto es el más experimentado. Para eso, puedo usar `.reduce()` también:

```
var mostExpPilot = pilots.reduce(function (oldest, pilot) {  
  return (oldest.years || 0) > pilot.years ? oldest : pilot;}, {});
```

Llamé a mi acumulador más antiguo. Mi callback de llamada compara el acumulador con cada piloto. Si un piloto tiene más años de experiencia que el más antiguo, entonces ese piloto se convierte en el nuevo más viejo, así que ese es el que esta dentro del return.

Como puedes ver, usar `.reduce ()` es una manera fácil de generar un único valor u objeto a partir de una matriz.

## Usando → `.filter()`

¿Qué sucede si tiene una matriz, pero solo quiere algunos de sus elementos?  
¡Ahí es donde entra en juego `.filter ()`!

Aquí están nuestros datos:

```
var pilots = [  
  { id: 2, name: "Wedge Antilles", faction: "Rebels", },  
  { id: 8, name: "Ciena Ree", faction: "Empire", },  
  { id: 40, name: "Iden Versio", faction: "Empire", },  
  { id: 66, name: "Thane Kyrell", faction: "Rebels", }];
```

Digamos que queremos dos conjuntos ahora: uno para los pilotos rebeldes, el otro para los imperiales. ¡Con `.filter ()` no podría ser más fácil!

```
var rebels = pilots.filter(function (pilot) {  
  return pilot.faction === "Rebels";});  
  
var empire = pilots.filter(function (pilot) {  
  return pilot.faction === "Empire";});
```

¡Eso es! Y es aún más corto con las arrow functions:

```
const rebels = pilots.filter(pilot => pilot.faction === "Rebels");  
const empire = pilots.filter(pilot => pilot.faction === "Empire");
```

## Combianando .map(), .reduce(), and .filter()

```
var personnel = [  
  { id: 5, name: "Luke Skywalker", pilotingScore: 98, shootingScore: 56,  
    isForceUser: true, },  
  { id: 82, name: "Sabine Wren", pilotingScore: 73, shootingScore: 99,  
    isForceUser: false, },  
  { id: 22, name: "Zeb Orellios", pilotingScore: 20, shootingScore: 59,  
    isForceUser: false, },  
  { id: 15, name: "Ezra Bridger", pilotingScore: 43, shootingScore: 67,  
    isForceUser: true, },  
  { id: 11, name: "Caleb Dume", pilotingScore: 71, shootingScore: 85,  
    isForceUser: true, },  
];
```

Nuestro objetivo: obtener la puntuación total de los usuarios de la fuerza solamente. ¡Hagámoslo paso a paso!

Primero, necesitamos filtrar al personal que no puede usar la fuerza:

```
var jediPersonnel = personnel.filter(function (person) {  
  return person.isForceUser;});  
  
// Result: [{...}, {...}, {...}] (Luke, Ezra and Caleb)
```

Con eso nos quedan 3 elementos en nuestra matriz resultante. Ahora necesitamos crear una matriz que contenga la puntuación total de cada Jedi.



```
var jediScores = jediPersonnel.map(function (jedi) {  
  return jedi.pilotingScore + jedi.shootingScore;}); // Result: [154, 110, 156]
```

Y usemos `.reduce()` para obtener el total:

```
var totalJediScore = jediScores.reduce(function (acc, score) {  
  return acc + score;}, 0); // Result: 420
```

Y ahora esta es la parte divertida ... podemos encadenar todo esto para obtener lo que queremos en una sola línea:

```
var totalJediScore = personnel .filter(function (person) {  
  return person.isForceUser; }) .map(function (jedi) {  
  return jedi.pilotingScore + jedi.shootingScore; }) .reduce(function (acc, score) {  
  return acc + score; }, 0);
```

Y ahora haciendo uso de las arrow functions:

```
const totalJediScore = personnel.filter(person => person.isForceUser)  
  .map(jedi => jedi.pilotingScore + jedi.shootingScore)  
  .reduce((acc, score) => acc + score, 0);
```

Boom! 💣

## Ejercicio

<https://gitlab.com/upgrade-hub/ejercicios-master/javascript/javascript-sesion-5>

**Continuamos con la sesión 6:**