

# OTROS MÉTODOS DE ARRAY

# isArray()

- Devuelve true si el objeto pasado por parámetro es un array, o false en caso contrario.

```
Array.isArray([1, 2, 3]);    // true  
Array.isArray({foo: 123});  // false  
Array.isArray("foobar");    // false  
Array.isArray(undefined);   // false
```

## every()

- Devuelve true si todos los elementos del array cumplen la función pasada por parámetro, o false si algún elemento no la cumple.

```
[12, 5, 8, 130, 44].every(elem => elem >= 10); // false  
[12, 54, 18, 130, 44].every(elem => elem >= 10); // true
```

## some()

- Devuelve true si algún elemento del array cumple la condición pasada por parámetro mediante una función, o false si ninguno la cumple.

```
[2, 5, 8, 1, 4].some(elem => elem > 10); // false  
[12, 5, 8, 1, 4].some(elem => elem > 10); // true
```

# filter()

- Devuelve un nuevo array con los elementos que cumplen la función pasada por parámetro.

```
function esSuficientementeGrande(elemento) {  
    return elemento >= 10;  
}  
var filtrados = [12, 5, 8, 130, 44].filter(esSuficientementeGrande);  
// filtrados es [12, 130, 44]
```

# forEach()

- Ejecuta la función pasada por parámetro para cada elemento del array.

```
var a = ["a", "b", "c"];  
  
a.forEach(function(element) {  
    | console.log(element);  
}); // a // b // c
```

# map()

- Devuelve un nuevo array con los resultados de aplicar la función pasada por parámetro a cada elemento del array.

```
var numbers = [1, 4, 9];  
var roots = numbers.map(Math.sqrt);  
// roots contiene [1, 2, 3]  
// numbers se mantiene [1, 4, 9]
```

# reduce()

- Reduce un array a un único valor, aplicando una función pasada por parámetro. El segundo parámetro opcional, indica el valor con el que se inicializará la variable acumuladora.
- El primer parámetro de dicha función almacena el valor devuelto por la ejecución anterior actuando como acumulador y el segundo hace referencia al elemento actual del array en la ejecución del método.

```
let sum = [65, 44, 12, 4].reduce((total, num) => total + num , 0);
```



# reduceRight()

- Similar a reduce() pero recorre el array en sentido inverso, de derecha a izquierda.

```
var flattened = [  
  [0, 1],  
  [2, 3],  
  [4, 5]  
].reduceRight(function(a, b) {  
  return a.concat(b);  
});  
// flattened = [4, 5, 2, 3, 0, 1]
```

# findIndex()

- Devuelve el índice del primer elemento que cumple la condición indicada, en caso contrario devuelve *-1*.
- Detecta el valor NaN.

```
var arr1 = ['a', NaN];  
arr1.findIndex(x => Number.isNaN(x)); // 1  
  
var arr2 = [3, 1, -1, -5];  
arr2.findIndex(x => x < 0); // 2
```

# find()

- Devuelve el valor del primer elemento que cumple la condición indicada, caso contrario devuelve *undefined*.
- Detecta el valor NaN.

```
const arr = ['a', NaN];  
console.log(arr.find(x => Number.isNaN(x))); //NaN  
  
[3, 1, -1, -5].find(x => x < 0); //-1
```

```
let users = [  
  { login: "Sam",    admin: false },  
  { login: "Brook",  admin: true  },  
  { login: "Tyler",  admin: true  }  
];  
  
let admin = users.find( (user) => {  
  return user.admin;  
});  
  
console.log( admin );
```

How can we find an admin in this array of users?

Returns first object for which user.admin is true

> { "login": "Brook", "admin": true }

## fill(valor, inicio, fin)

- Permite rellenar elementos de un array con un mismo valor y en las posiciones definidas.

```
var arr1 = new Array(3).fill('a'); // ['a', 'a', 'a']  
  
var arr2 = ['a', 'b', 'c', 'd'];  
arr2.fill(null, 2, 3); // ['a', 'b', null, 'd']
```

## copyWithin(índice,inicio,fin)

- Copia los elementos comprendidos entre inicio y fin, a partir de la posición indicada (índice).
- Por defecto '*inicio = 0*' y '*fin = length*'.
- Sobrescribe las posiciones afectadas.

```
var arr = [0, 1, "x", "y", 4, 5, 6];  
arr.copyWithin(3, 2, 4); // [0, 1, "x", "x", "y", 5, 6]
```

# includes(elemento, inicio)

- Devuelve true o false si el elemento está incluido en el array.

```
[1, 2, 3].includes(2);    // true
[1, 2, 3].includes(4);    // false
[1, 2, 3].includes(3, 3); // false
[1, 2, 3].includes(3, -1); // true
[1, 2, NaN].includes(NaN); // true
```

## of(element0[, element1[, ...[, elementN]])

- Crea un array con los elementos pasados por parámetros.

```
var arr1 = Array.of(1);           // [1]
var arr2 = Array.of(1, 2, 3);     // [1, 2, 3]
var arr3 = Array.of(undefined);  // [undefined]
```

## from(obj, mapFn, thisArg)

- Crea una nueva instancia de Array a partir de un array-like o un objeto iterable(Maps, Sets, strings).



```
//Array de un array-like
var arrayLike = {length: 2, 0: 'x', 1: 'y'};

var array = Array.from(arrayLike); //ahora es iterable
for (var elemento of array) {
  console.log(elemento);
}
// x // y

//Array de un String
Array.from("foo"); // ["f", "o", "o"]

//Array de un Set
var s = new Set(["foo", window]);
Array.from(s); // ["foo", window]

//Array from a Map
var m = new Map([
  [1, 2],
  [2, 4],
  [4, 8]
]);
Array.from(m); // [[1, 2], [2, 4], [4, 8]]

//Array.from() + funcion flecha
Array.from([1, 2, 3], x => x + x);
// [2, 4, 6]
```

# keys()

- Nos devuelve un nuevo objeto Array Iterator que contiene las claves por cada índice del array.

```
var arr = ['x', 'y', 'z'];  
  
for (var key of arr.keys()) {  
  |   console.log(key);  
}  
//0  
//1  
//2
```

# values()

- Nos devuelve un nuevo objeto Array Iterator que contiene los valores por cada índice del array.

```
var arr = ['x', 'y', 'z'];
var valores = arr.values();
while (true) {
    var elemento = valores.next();
    if (elemento.done)
        break;
    console.log(elemento.value);
}
// "x"
// "y"
// "z"
```

# entries()

- Nos devuelve un nuevo objeto Array Iterator que contiene las parejas clave/valor por cada índice del array.

```
var a = ['a', 'b', 'c'];  
var iterator = a.entries();  
  
console.log(iterator.next().value); // [0, 'a']  
console.log(iterator.next().value); // [1, 'b']  
console.log(iterator.next().value); // [2, 'c']
```

# Anexo

- Algunos métodos de Array que se definieron en ES5 ignoran los huecos en arrays, mientras que otros los eliminan, y otros los consideran elementos undefined.
- Los métodos que añade ES6, tratan siempre los huecos de array como elementos undefined.

# Anexo

- ES6 nos permite acceder al prototype de Array a través de un Array vacío literal, es decir, [].

```
//ES5  
Array.prototype.slice.call(arguments)  
  
//ES6  
[].slice.call(arguments)
```

# Ejercicio 1

- Crea un array de diez números aleatorios entre 0 y 10.
- Devuelve un array con los valores menores a 8.
- Intercambia los valores impares por la cadena “impar”.

## Ejercicio 2

- Crea un vector con 5 valores numéricos.
- Comprueba si son todos enteros positivos.
- Devuelve en un nuevo array el triple sus valores.
- Calcula el resultado de multiplicar todos los valores del nuevo array.



## Ejercicio 3

- Crea un array que contenga al menos 5 tweets.
  1. La primera letra de cada tweet debe estar en mayúscula.
  2. Devolver en un nuevo array las longitudes de cada tweet.
  3. Comprobar si el usuario habla en alguno de sus tweets sobre 'software'.
  4. En caso afirmativo, muéstralo por consola
  5. Comprobar si todas las entradas cumplen la antigua restricción de Twitter (inferior a 140 caracteres)
  6. ¿Cuántos caracteres ha escrito el usuario de media?