

Metodos Array

Avanzados

```
var newArray = arr.filter(callback(currentValue[, index[,  
array]]))
```

Filter *sintáxis general*

- Filtra un array, en base a una condición (dada por una función).

- Ejemplo de condición:

```
function menosDe50(num)
{
    return num < 50;
}
```

- Devuelve un array diferente (no modifica el original).

- Ejemplo de uso:

```
function filtra(array)
{
    return array.filter(menosDe50);
}
```

Filter

- Forma “tradicional”.

```
function filtra(arr)
{
    var arrAux;
    for(let i = 0; i < arr.length; i++)
    {
        if(arr[i]>50)
        {
            arrAux.push(arr[i]);
        }
    }
}
```

Filter

```
var nuevo_array = arr.map(function  
  callback(currentValue, index, array) { // Elemento  
    devuelto de nuevo_array })
```

map sintáxis general

- Devuelve un array con el resultado de llamar a una función en cada valor.

```
function cuadrado(arr)
{
    return arr.map(Math.sqrt);
}
```

- Tradicional:

```
function raiz(arr)
{
    var arrAux;
    for(let i = 0; i < arr.length; i++)
    {
        arrAux.push(Math.sqrt(arr[i]));
    }
}
```

Map

```
arr.reduce(callback(acumulador, valorActual[, índice[,  
array]])([, valorInicial])
```

reduce sintáxis general

- Reduce todos los valores del array a un único valor.

- Definición de la operación:

```
function sumaTotal(total, num)
{
    return total + num;
}
```

- Llamada al método.

```
array.reduce(sumaTotal);
```

Reduce

- Método “tradicional”.

```
function sumaArray(arr)
{
    var total=0;
    for(let i = 0; i < arr.length; i++)
    {
        total+=arr[i];
    }
    return total
}
```

Reduce

```
arr.findIndex(callback( element[, index[, array]] )[, thisArg])
```

Element El elemento actual siendo procesado en el array.

Index Optional

El índice del elemento actual que está siendo procesado en el array

Array Optional

El array `findIndex` de donde fue llamado.

findIndex *sintáxis general*

El método **findIndex()** devuelve el índice del primer elemento de un array que cumpla con la condición de la función proporcionada.

La función **findIndex** es parecida a **indexOf**, con un par de diferencias:

- Le pasamos como argumento una *arrow function* o una function. Es la condición que tiene que cumplir el elemento del array que queremos detectar.
- Permite **detectar NaN**

findIndex

```
var array1 = [5, 12, 8, 130, 44];

function findFirstLargeNumber(element) {
  return element > 13;
}

console.log(array1.findIndex(findFirstLargeNumber));
// expected output: 3
```

firstIndex

```
arr.find(callback(element[, index[, array]])([, thisArg])
```

Find *sintáxis general*

El método **find()** devuelve el **valor** del **primer elemento** del array que cumple la función de prueba proporcionada. En cualquier otro caso se devuelve undefined.

```
const array = [14, 123, 50, 20, 312];  
  
const myVal = array.find(function(element) {  
  return element > 100;  
});  
  
//Output: 123
```

find

- Todos los ejemplos indistintamente se pueden realizar indistintamente con funciones anónimas o bien con funciones convencionales.

- **fill**, crea un Array de un tamaño determinado e inicializar su contenido.

```
const myArray = new Array(size).fill(value, start?, end?);
```

```
const array = new Array(3).fill('a'); //[ 'a', 'a', 'a' ]
```

```
[ 'a', 'b', 'c', 'd' ].fill(null, 2, 3); //[ 'a', 'b', null, 'd' ]
```

Fill ***arr.fill(value[, start = 0[, end = this.length]])***