

# JavaScript

## FILTROS



AÑADIR



ELIMINAR



EXTRAER



## TRANSFORMAR



COMPROBACIONES



ACUMULADORES



## SEPARAR Y UNIR



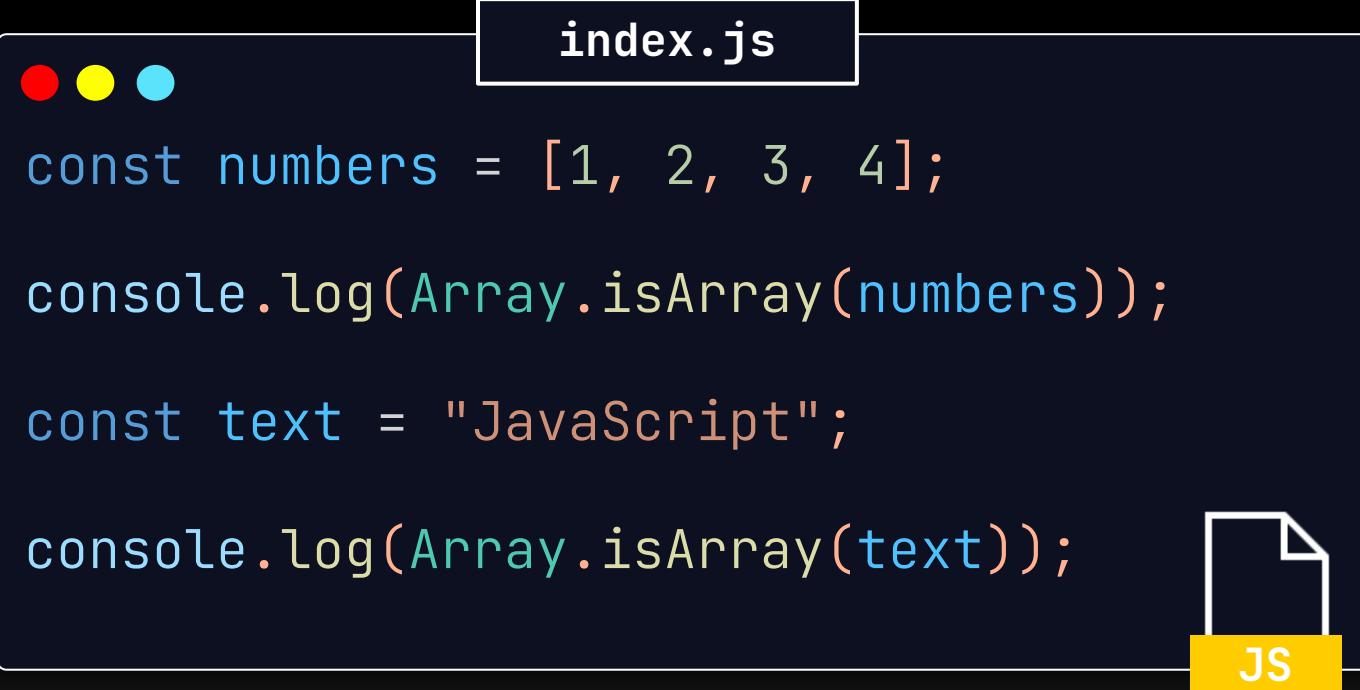
BÚSQUEDA



## ESTATICOS

- `Array.isArray()`
- `Array.of()`

El método estático `Array.isArray()` verifica si el argumento pasado es una matriz o no.



index.js

```
const numbers = [1, 2, 3, 4];  
  
console.log(Array.isArray(numbers));  
  
const text = "JavaScript";  
  
console.log(Array.isArray(text));
```

JS



TERMINAL

```
$ node index.js  
true  
false
```



El método Array.***of()*** crea una instancia de Array (la diferencia con el constructor es como maneja los parámetros de tipo entero).



### index.js

```
const arr1 = Array.of(7);
console.log(arr1);

const arr2 = Array.of(1, 2, 3);
console.log(arr2);

const arr3 = Array(7);
console.log(arr3);

const arr4 = Array(1, 2, 3);
console.log(arr4);
```



### TERMINAL

```
$ node index.js
[7]
[1, 2, 3]
[undefined, undefined,
undefined, undefined,
undefined, undefined,
undefined]
[1, 2, 3]
```



El método `.concat()` devuelve la unión de dos o más matrices (no modifica las matrices involucradas).

### index.js

```
● ● ●  
const heroesDC = [  
  "superman", "aquaman", "batman", "flash"  
];  
  
const heroesMarvel = [  
  "deadpool", "hulk", "wolverine", "ironman"  
];  
  
const multiverso = heroesDC.concat(  
  heroesMarvel);  
  
multiverso.forEach(hero => {  
  console.log(hero)  
});
```



### TERMINAL

```
● ● ●  
$ node index.js  
"superman"  
"aquaman"  
"batman"  
"flash"  
"deadpool"  
"hulk"  
"wolverine"  
"ironman"
```



El método `.every()` devuelve `true` si todos los elementos cumplen la condición en la función.

### index.js

```
● ● ●  
const arrNumbers1 = [10, 20, 55.33, 23];  
const arrNumbers2 = [2, 4, 6, 8, 10, 12];  
  
function numberIsEven(num) {  
  return Math.round(num) % 2 === 0;  
}  
  
console.log(arrNumbers1.every(numberIsEven))  
;  
  
console.log(arrNumbers2.every(numberIsEven))  
;
```



### TERMINAL

```
● ● ●  
$ node index.js  
false  
true
```



El método `.entries()` devuelve un objeto [Array Iterator] con pares clave/valor

### index.js

```
const numbers = [10, 20, 55.33, 23];  
  
const n = numbers.entries();  
  
for (let x of n) {  
  console.log(x);  
}
```



### TERMINAL

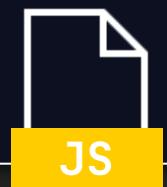
```
$ node index.js  
[0, 10]  
[1, 20]  
[2, 55.33]  
[3, 23]
```



El método `.filter()` crea una nueva matriz con los elementos que cumplen una condición.

### index.js

```
● ● ●  
const numbers = [10, 20, 55.33, 23];  
  
function numberIsEven(num) {  
  return Math.round(num) % 2 === 0;  
}  
  
const pares = numbers.filter(numberIsEven);  
console.log(pares);
```



### TERMINAL

```
● ● ●  
$ node index.js  
[10, 20]
```



El método `.find()` devuelve el valor del primer elemento que cumple una condición.



### index.js

```
const heroes = [  
  "superman",  
  "aquaman",  
  "batman",  
  "ironman"  
];  
  
const batman = heroes.find(hero => {  
  return hero === "batman"  
});  
  
console.log(batman);
```



### TERMINAL

```
$ node index.js  
"batman"
```



El método `.findLast()` devuelve el valor del elemento que cumple una condición, pero buscando elementos desde derecha a izquierda.



### index.js

```
const heroes = [  
  "superman",  
  "aquaman",  
  "batman",  
  "ironman",  
  "batwoman"  
];  
  
const heroe = heroes.findLast(hero => {  
  return hero.startsWith("b");  
});  
  
console.log(hero);
```



### TERMINAL

```
$ node index.js  
"batwoman"
```



El método `.findIndex()` devuelve el valor de la posición en la matriz del primer elemento que cumple una condición.



### index.js

```
const heroes = [  
  "superman",  
  "aquaman",  
  "batman",  
  "ironman"  
];  
  
const index = heroes.findIndex(hero => {  
  return hero === "batman"  
});  
  
console.log(index);
```



### TERMINAL

```
$ node index.js  
2
```



El método `.findIndexLast()` devuelve el valor de la posición en la matriz del elemento que cumple una condición, pero buscando desde derecha a izquierda.

### index.js

```
● ● ●
const heroes = [
  "superman",
  "aquaman",
  "batman",
  "ironman",
  "batwoman"
];

const index = heroes.findIndex(hero => {
  return hero.startsWith("b");
});

console.log(index);
```



### TERMINAL

```
$ node index.js
4
```

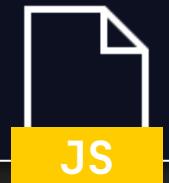


El método `.shift()` elimina el primer elemento de la matriz y cambia todos los demás elementos a un índice más bajo.



### index.js

```
const numbers = [10, 20, 55.33, 23];  
  
numbers.shift();  
  
numbers.forEach((num, index) => {  
  console.log(`${index}, ${num}`)  
});
```



### TERMINAL

```
$ node index.js  
"0, 20"  
"1, 55.33"  
"2, 34"
```



El método `.some()` devuelve `true` si al menos uno de los elementos cumplen la condición en la función callback.



### index.js

```
const arrNumbers1 = [10, 20, 55.33, 23];
const arrNumbers2 = [1, 3, 5, 9, 11, 13];

function numberIsEven(num) {
  return Math.round(num) % 2 === 0;
}

console.log(arrNumbers1.some(numberIsEven));
console.log(arrNumbers2.some(numberIsEven));
```



### TERMINAL

```
$ node index.js
true
false
```



El método `.unshift()` agrega nuevos elementos al comienzo de la matriz (sobrescribe la original).



index.js

```
const heroes = [  
  "superman",  
  "aquaman",  
  "batman",  
  "ironman"  
];  
  
heroes.unshift("flash", "spider-man");  
  
heroes.forEach(hero => {  
  console.log(hero)  
});
```



TERMINAL

```
$ node index.js  
"flash"  
"spider-man"  
"superman"  
"aquaman"  
"batman"  
"ironman"
```



El método `.sort()` ordena los elementos como cadenas en orden alfabético y ascendente



index.js

```
const heroes = [  
  "superman",  
  "aquaman",  
  "batman",  
  "ironman"  
]
```

```
heroes.sort()  
.forEach(hero => {  
  console.log(hero)  
});
```



TERMINAL

```
$ node index.js  
"aquaman"  
"batman"  
"ironman"  
"superman"
```



El método `.reduce()` devuelve un único valor; el resultado acumulado de la función reductora.



### index.js

```
const numbers = [10, 20, 55.33, 23];  
  
function getSum(total, num) {  
  return total + Math.round(num);  
}  
  
const total = numbers.reduce(getSum, 0);  
console.log(total);
```



### TERMINAL

```
$ node index.js  
108
```



El método `.reduceRight()` devuelve un único valor; el resultado acumulado de la función reductora (acumulando el valor de derecha a izquierda).

### index.js



```
const numbers = [95, 5, 25, 10, 25];

const result = numbers.reduceRight((first, second) => {
  console.log(`F=${first} S=${second}`);
  return first - second;
});

console.log(result);
```



### TERMINAL



```
$ node index.js
  "F=25 S=10"
  "F=15 S=25"
  "F=-10 S=5"
  "F=-15 S=95"
  - 110
```



El método `.pop` elimina un elemento de la matriz (el del final, devuelve el elemento eliminado).



### index.js

```
const heroesDC = [  
  "batgirl",  
  "green arrow",  
  "catwoman",  
  "shazam"  
];  
  
console.log(heroesDC.pop());  
  
heroesDC.forEach(hero =>  
  console.log(hero));
```



### TERMINAL

```
$ node index.js  
"shazam"  
  
"batgirl"  
"green arrow"  
"catwoman"
```



El método `.push()` agrega un nuevo elemento a una matriz  
(al final, sobrescribe la original).



### index.js

```
const heroesDC = [  
  "batgirl",  
  "green arrow",  
  "catwoman",  
  "shazam"  
];  
  
heroesDC.push("lterna verde",  
  "mujer maravilla");  
  
heroesDC.forEach(hero =>  
  console.log(hero));
```



### TERMINAL



```
$ node index.js  
"batgirl"  
"green arrow"  
"catwoman"  
"shazam"  
"lterna verde"  
"mujer maravilla"
```



El método `.splice()` se utiliza para agregar nuevos elementos a una matriz (indicando su posición y cantidad de elementos a eliminar) (modifica la matriz original).

### index.js

```
● ● ●  
const heroesDC = [  
  "batgirl",  
  "green arrow",  
  "catwoman",  
  "shazam"  
];  
  
heroesDC.splice(2, 1,  
  "laterna verde", "mujer maravilla")  
  
heroesDC.forEach(hero =>  
  console.log(hero));
```



JS

### TERMINAL

```
● ● ●  
$ node index.js  
"batgirl"  
"green arrow"  
"laterna verde"  
"mujer maravilla"  
"shazam"
```



El método `.slice()` se utiliza para extraer parte de una matriz, toma dos argumentos la posición de un elemento de inicio y hasta un elemento final (no incluye el elemento final).

### index.js

```
● ● ●  
const heroes = [  
  "batgirl",  
  "green arrow",  
  "catwoman",  
  "shazam",  
  "capitán américa",  
  "iron man",  
  "batman"  
];  
  
const heroesMarvel = heroes.slice(4, 6);  
  
heroesMarvel.forEach(hero =>  
  console.log(hero));
```



### TERMINAL

```
● ● ●  
$ node index.js  
"capitán américa"  
"iron man"
```



El método `.map()` se utiliza para transformar una matriz, llama una función una vez para cada elemento de la matriz y así crea una matriz nueva (no modifica la matriz original, no ejecuta la función en elementos vacíos).

### index.js

```
● ● ●  
const heroes = [  
    "batgirl",  
    "green arrow",  
    "catwoman",  
    "shazam",  
    "capitán américa",  
    "iron man",  
    "batman"  
];  
  
const longitudNombres = heroes.map((nombre) =>  
    nombre.length  
);  
  
for (let n of longitudNombres) {  
    console.log(parseInt(n));  
}
```



### TERMINAL

```
$ node index.js  
7  
11  
8  
6  
15  
8  
6
```



El método `.join()` se utiliza para crear un `string` con todos los elementos de una matriz, uniéndolo por el texto que le pasemos por parámetro.



index.js

```
const letras = ["a", "b", "c", "d"];  
  
const unidosPorPunto = letras.join(".");  
const unidosPorFlecha = letras.join("->");  
  
console.log(unidosPorPunto);  
console.log(unidosPorFlecha);
```



TERMINAL

```
$ node index.js  
"a.b.c.d"  
"a->b->c->d"
```



El método `.split()` de un `string` es posible crear una matriz, separándolo por el texto que le pasemos por parámetro (separador).

### index.js

```
● ● ●  
const splitLetras = "a.b.c".split(".");
const splitDigitos = "5-4-3-2-1".split("-");

console.log(splitLetras);
console.log(splitDigitos);
```



JS

### TERMINAL

```
● ● ●  
$ node index.js
["a", "b", "c"]
["5", "4", "3", "2", "1"]
```

