Índice



## 1. Crear vectores.

Con new:

```
let a = new Array(); // Crea un array vacío
a[0] = 13;
console.log(a[0]); // Imprime 13
```

Con []:

```
let a = ["a", "b", "c", "d", "e"]; // Array de tamaño 5, con 5 valores inicialmente
console.log(a); // Imprime ["a", "b", "c", "d", "e"]
```

## 2. Propiedad lenght.

La longitud de un array depende de las posiciones que han sido asignadas:

```
let a = new Array(12); // Crea un array de tamaño 12
console.log(a.length); // Imprime 12
```

```
let a = ["a", "b", "c", "d", "e"]; // Array con 5 valores
console.log(a.length); // Imprime 5
```

## 3. Insertar valores: unshift y push.

unshift insertar valores al principio de un array y push al final:

```
let a = [];
a.push("a"); // Inserta el valor al final del array
a.push("b", "c", "d"); // Inserta estos nuevos valores al final
console.log(a); // Imprime ["a", "b", "c", "d"].
a.unshift("A", "B", "C"); // Inserta nuevos valores al principio del array
console.log(a); // Imprime ["A", "B", "C", "a", "b", "c", "d"].
```

## 4. Eliminar valores: shift y pop.

**shift** elimina del principio y **pop** del final () del array. Estas operaciones nos devolverán el valor que ha sido eliminado.

```
let a = ["A", "B", "C", "a", "b", "c", "d"];
console.log(a.pop()); // Imprime y elimina la última posición → "d"
console.log(a.shift()); // Imprime y elimina la primera posición → "A"
console.log(a); // Imprime ["B", "C", "a", "b", "c"]
```

## 5. Imprimir vectores: join y toString.

toString():

```
let a = [3, 21, 15, 61, 9];
console.log(a.toString());// Imprime "3,21,15,61,9"
```

**join**(). Por defecto, devuelve un string con todos los elementos separados por coma. Sin embargo, podemos especificar el separador a imprimir.

```
let a = [3, 21, 15, 61, 9];
console.log(a.join()); // Imprime "3,21,15,61,9", igual que toString()
console.log(a.join(" -#- ")); // Imprime "3 -#- 21 -#- 15 -#- 61 -#- 9"
```

## 6. Concatenar: concat.

Concat:

```
let a = ["a", "b", "c"];
let b = ["d", "e", "f"];
```

```
let c = a.concat(b);
console.log(c); // Imprime ["a", "b", "c", "d", "e", "f"]
console.log(a); // Imprime ["a", "b", "c"] . El array a no ha sido modificado
```

## 7. Trocear vectores: slice y splice

**slice** nos devuelve un nuevo array a partir de posiciones intermedias de otro.

```
let a = ["a", "b", "c", "d", "e", "f"];
let b = a.slice(1, 3); // (posición de inicio → incluida, posición final → excluida)
console.log(b); // Imprime ["b", "c"]
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]. El array original no es modificado
console.log(a.slice(3)); // Un parámetro. Devuelve desde la posición 3 al final → ["d", "e", "f"]
```

**splice** elimina elementos del array original y devuelve los elementos eliminados. También permite insertar nuevos valores

```
let a = ["a", "b", "c", "d", "e", "f"];
a.splice(1, 3); // Elimina 3 elementos desde la posición 1 ("b", "c", "d")
console.log(a); // Imprime ["a", "e", "f"]
a.splice(1,1, "g", "h"); // Elimina 1 elemento en la posición 1 ("e"), e inserta "g", "h" en esa posición
console.log(a); // Imprime ["a", "g", "h", "f"]
a.splice(3, 0, "i"); // En la posición 3, no elimina nada, e inserta "i"
console.log(a); // Imprime ["a", "g", "h", "i", "f"]
```

## 8. Ordenar vectores: reverse y sort

**Reverse:** invertir el orden del array.

```
let a = ["a", "b", "c", "d", "e", "f"];
a.reverse(); // Hace el reverse del array original
console.log(a); // Imprime ["f", "e", "d", "c", "b", "a"]
```

**sort:** ordenar los elementos de un array .

```
let a = ["Peter", "Anne", "Thomas", "Jen", "Rob", "Alison"];
a.sort(); // Ordena el array original
console.log(a); // Imprime ["Alison", "Anne", "Jen", "Peter", "Rob", "Thomas"]
```

Ordenar vectores con elementos distintos de string:

```
let a = [20, 6, 100, 51, 28, 9];
a.sort(); // Ordena el array original
console.log(a); // Imprime [100, 20, 28, 51, 6, 9]

a.sort(function(n1, n2) {
return n1 - n2;
});
console.log(a); // Imprime [6, 9, 20, 28, 51, 100]
```

## 9. Buscar elementos: indexOf y lastIndexOf.

**indexOf: p**ermite conocer si el valor que le pasamos se encuentra en el array o no. Si lo encuentra nos devuelve la primera posición donde está, y si no, nos devuelve -1.

```
let a = [3, 21, 15, 61, 9, 15];
console.log(a.indexOf(15)); // Imprime 2
console.log(a.indexOf(56)); // Imprime -1. No encontrado
```

**lastIndexOf** nos devuelve la primera ocurrencia encontrada empezando desde el final.

```
let a = [3, 21, 15, 61, 9, 15];
console.log(a.lastIndexOf(15)); // Imprime 5
```

## 10. Buscando elementos que cumplen condiciones: every y some.

**every** devolverá un boolean indicando si todos los elementos del array cumplen cierta condición. Esta función recibirá cualquier elemento, lo testeará, y devolverá cierto o falso dependiendo de si cumple la condición o no.

```
let a = [3, 21, 15, 61, 9, 54];
console.log(a.every(function(num) { // Comprueba si cada número es menor a 100
return num < 100;
})); // Imprime true
console.log(a.every(function(num) { // Comprueba si cada número es par
return num % 2 == 0;
})); // Imprime false
```

**some** es similar a every, pero devuelve cierto en el momento en el que uno de los elementos del array cumple la condición.

```
let a = [3, 21, 15, 61, 9, 54];
console.log(a.some(function(num) { // Comprueba si algún elemento del array es par
return num % 2 == 0;
})); // Imprime true
```

## 11. Iterar con forEach.

Podemos iterar por los elementos de un array usando el método **forEach**. De forma opcional, podemos llevar un seguimiento del índice al que está accediendo en cada momento, e incluso recibir el array como tercer parámetro.

Es importante recalcar que si se modifican los elementos de un array en un foreach los cambios no se guardan en el array, es decir, no podemos modificar el propio array dentro de un foreach.

```
let a = [3, 21, 15, 61, 9, 54];
let sum = 0;
a.forEach(function(num) { //
sum += num;
});
console.log(sum); // Imprime 163
a.forEach(function(num, indice, array) { // índice y array son parámetros opcionales
console.log("Índice " + indice + " en [" + array + "] es " + num);
}); // Imprime -> Índice 0 en [3,21,15,61,9,54] es 3, Índice 1 en [3,21,15,61,9,54] es
21, ...
```

## 12. Modificar todos los elementos de un array: map.

El método **map** recibe una función que transforma cada elemento y lo devuelve. Este método devolverá al final un nuevo array del mismo tamaño conteniendo todos los elementos resultantes.

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.map(function(num) {
return num*2;
})); // Imprime [8, 42, 66, 24, 18, 108]
```

## 13. Filtrar los elementos de un array: filter.

**filter** filtra los elementos de un array y obtiene como resultado un array que contiene sólo los elementos que cumplan cierta condición.

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.filter(function(num) {
return num % 2 == 0; // Si devuelve true, el elemento se queda en el array devuelto
})); // Imprime [4, 12, 54]
```

## 14. Acumular valores: reduce y reduceRight

**reduce** usa una función que acumula un valor, procesando cada elemento (segundo parámetro) con el valor acumulado (primer parámetro):

```
let a = [4, 21, 33, 12, 9, 54];
console.log(a.reduce(function(total, num) { // Suma todos los elementos del array
return total + num;
```

```
}, 0)); // Imprime 133
console.log(a.reduce(function(max, num) { // Obtiene el número máximo del array
return num > max? num : max;
}, 0)); // Imprime 54
```

**reduceRight** hace lo mismo que reduce pero al revés:

```
let a = [4, 21, 33, 12, 9, 154];
console.log(a.reduceRight(function(total, num) { // Comienza con el último número y
resta todos los otros
números
return total - num;
})); // Imprime 75 (Si no queremos enviarle un valor inicial, empezará con el valor de
      la última posición del array
```

## 15. Dividir cadenas: split

Esta función sirve para dividir una cadena en partes utilizando un carácter delimitador, devolviéndonos un array con los "trozos". También admite un segundo parámetro opcional que indica cuántos elementos queremos que nos devuelva.

## 16. Formato JSON.

El formato JSON (https://es.wikipedia.org/wiki/JSON) nos permite definir objetos y array de objetos en JavaScript. Más adelante veremos como trabajar con ficheros en formato JSON por ahora nos interesa este formato para vectores, por ejemplo si definimos:

```
let datos = [
{nombre: "Nacho", telefono: "966112233", edad: 40},
{nombre: "Ana", telefono: "911223344", edad: 35},
{nombre: "Mario", telefono: "611998877", edad: 15},
{nombre: "Laura", telefono: "633663366", edad: 17}
];
```

Tenemos un array llamado datos con 4 posiciones. Para acceder a una posición del array: datos[0]. Y para acceder a una propiedad concreta dentro de una posición: datos[0].edad.

Además podemos usar cualquier método de los vistos anteriormente con array en formato json.