Contenido

1. Crear vectores.

Con new:

```
let a = new Array (); // Crear un vector vacío
a [0] = 13;
console.log (a[0]); // Print 13
let b = new Array (2); // Crear un vector de 2 elementos pero sin valores dentro
```

Con []:

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

2. Rellenar un array: fill

El método fill() en JavaScript se utiliza para llenar todos los elementos de un array con un valor específico. Este método sobrescribe los valores existentes en el array, asignando el mismo valor a cada posición.

```
const arr = [1, 2, 3, 4, 5];
```

```
arr.fill(0); // Llena todo el array con 0
console.log(arr); // [0, 0, 0, 0, 0]
```

3. Propiedad lenght.

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

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


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

unshift() → agrega al princ

shift() → elimina del princi

push() → agrega al final.

pop() → elimina del final.

4. Insertar valores: unshift y push.

unshift inserta 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); // Print ["a", "b", "c", "d"].
a.unshift ("A", "B", "C"); // Inserta los nuevos valores al principio del array
console.log (a); // Print ["A", "B", "C", "a", "b", "c", "d"].
```

5. Borrar valores: shift y pop.

shift borra del principio y **pop** del final del array. Esas operaciones devolverán el valor que ha sido borrado.

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

6. Imprimir vectores: join y toString.

toString ():

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

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

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

7. Concatenar: concat.

OJO Para concatenar strings basta con usar +
concat une arrays sin modificar el original
OJO porque objetos no los une, copia la referencia al objeto. Si lo modificas, lo modificas en el original (apuntan al mismo sitio en la memoria

Concat:

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

8. Vectores de corte: slice and splice

slice devuelve un nuevo array de posiciones intermedias de otro. El array original NO se modifica

```
let a = ["a", "b", "c", "d", "e", "f"];
let b = a.slice (1, 3); // (posición inicial → incluida, posición final → excluida)
console.log (b); // Print ["b", "c"]
console.log (a); // Print ["a", "b", "c", "d", "e", "f"]. El vector original no se
modifica
console.log (a.slice (3)); // Devuelve de la posición 3 hasta el final → ["d", "e", "f"]
```

splice elimina los elementos del vector original y devuelve los elementos borrados. También permite insertar nuevos valores. El array original se modifica

```
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); // Print ["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); // Print ["a", "g", "h", "f"]
a.splice (3, 0, "i"); // En la posición 3, no elimina nada, e inserta "i"
console.log (a); // Print ["a", "g", "h", "i", "f"]
```

1 argumento, indica la posición donde quieres modificar el array. 2º posición cuántos elementos quieres borrar si pones 0 no borras nada, y tercer argumento y siguientes se añade en la posición empezaste. Si quieres añadir en otro sitio tendrías que usar dos splice

9. Vectores de orden: reverse y sort

Reverse: revierte el orden de un array.

```
let a = ["a", "b", "c", "d", "e", "f"];
a.reverse (); // Revierte el vector original
console.log (a); // Print ["f", "e", "d", "c", "b", "a"]
```

sort: ordena los elementos de un array.

```
let a = ["Peter", "Anne", "Thomas", "Jen", "Rob", "Alison"];
```

```
a.sort (); // Ordena el vector original
console.log (a); // Print ["Alison", "Anne", "Jen", "Peter", "Rob", "Thomas"]
```

Ordena vectores con otros elementos distintos de string:

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

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

sort ordena usando unicode, por eso 100 va delante de 20.
Para evitar esto, usas sort con una función.

```
const personas = [
  { nombre: "Ana", edad: 30 },
  { nombre: "Luis", edad: 25 },
  { nombre: "Carlos", edad: 40 }
];

personas.sort(function(a, b) {
  return a.edad - b.edad; // Ordena de menor a mayor edad
});
```

sort acepta por parámetro una comparación, toma dos elementos del array y se los pasa a la función. La función consigue un número negativo, positivo o 0 y sort ya sabe qué hacer con ese núm
Compara pares e interpreta el resultado

10. Busca elementos: indexOf y lastIndexOf.

indexOf: Nos permite saber si el valor que le pasamos está en el array o no. Si lo encuentra, devuelve la primera posición donde está, y si no, devuelve -1

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

lastIndexOf devuelve la primera ocurrencia encontrada empezando por el final.

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

11.Buscando elementos que cumplan las condiciones: every y some.

Every devolverá un boolean indicando si todos los elementos del array cumplen la condición. Esta función recibirá todos los elementos, los probará y devolverá true o false dependiendo de si cumple la condición o no.

some es similar a every, pero devuelve true cuando uno de los elementos del vector cumple la condición.

```
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

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
```

every internamente recorre el array y en cada paso toma un elemento, se lo pasa a la función como argumento y evalúa si cumple la condición.

12. Iterar con forEach.

Podemos iterar a través de los elementos de una matriz usando el método forEach. Opcionalmente, podemos hacer un seguimiento del índice al que se accede 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 array mismo 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
```

for each permite ejecutar 1 función para cada elemento de la lista pero NO devuelve resultado. Hace acciones.

```
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, ...
```

aquí la función for each recibe 3 parámetros: num es el valor, índice el índice, array el array completo

13. Modificar 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]
```

```
const personas= [ {nombre: "Ana", edad:25},
{nombre: "luis", edad:30 }];

console.log(personas.map(function(persona) {
  return persona.nombre;
}));
```

14. Filtra los elementos de un array: filter.

filter filtra los elementos de un array y devuelve un array que contiene solo los elementos que cumplen la 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]
```

15. Acumular valores: reduce y reduceRight

reduce recibe dos cosas: la función reductora y un valor inicial opcional. Este valor inicial es el primer valor con que comienza el acumulador (acum).
Si no se pasa valor inicial, el acumulador inicia con el primer elemento del array y la iteración comienza desde el segundo elemento. Por eso ponemos 0))

reduce usa una función que acumula un valor, y procesa 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
```

16. Dividir strings: split

Esta función es usada para dividir un string en partes usando un carácter delimitador, devolviendo un array con los "trozos". También admite un segundo parámetro opcional que indica cuantos elementos queremos que devuelva.

```
let mensaje = 'Soy un tipo feliz';

// Dividiendo la cadena "mensaje" usando el carácter espacio
let arr = mensaje.split(' ');

// El arreglo
console.log(arr); // ["Soy", "un", "tipo", "feliz"]


// Acceso a cada elemento del arreglo resultante
console.log(arr[0]); // "Soy"
console.log(arr[1]); // "un"
console.log(arr[2]); // "tipo"
console.log(arr[3]); // "feliz"


console.log(mensaje.split(' ', 2)); // ["Soy", "un"]
```

Devuelve 2 elementos

17. Verificar si contiene un elemento: includes

El método includes() en JavaScript se utiliza para verificar si un array contiene un elemento específico. Retorna true si el elemento está presente, y false si no lo está. Es una forma sencilla y eficiente de comprobar si un valor ya existe en un array sin necesidad de recorrerlo manualmente.

```
const numeros = [1, 2, 3, 4, 5];
console.log(numeros.includes(3)); // true (el número 3 está en el array)
console.log(numeros.includes(6)); // false (el número 6 no está en el array)
```

OJO no sirve para objetos, sino para strings o números

18. Formato JSON.

El formato JSON (https://es.wikipedia.org/wiki/JSON) nos permite definir objetos y objetos de arrray en JavaScript. Después veremos cómo trabajar con ficheros en formato JSON. Por ahora estamos interesados en este formato para vector, por ejemplo, podemos definir:

```
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 un propiedad especifica dentro de una posición: datos [0] .edad.

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

```
const personas = [
  { nombre: "Ana", edad: 30 },
  { nombre: "Luis", edad: 25 },
  { nombre: "Carlos", edad: 40 }
];

personas.sort(function(a, b) {
  return a.edad - b.edad; // Ordena de menor a mayor
edad
});
```