

Estudiaremos las estructuras de datos organizados en colecciones que pueden manipularse como si fueran una única unidad, que a su vez se combina con otros para crear estructuras más complejas.

Arrays, Sets y Maps son las principales colecciones utilizadas en JavaScript.

## 4.1. Arrays

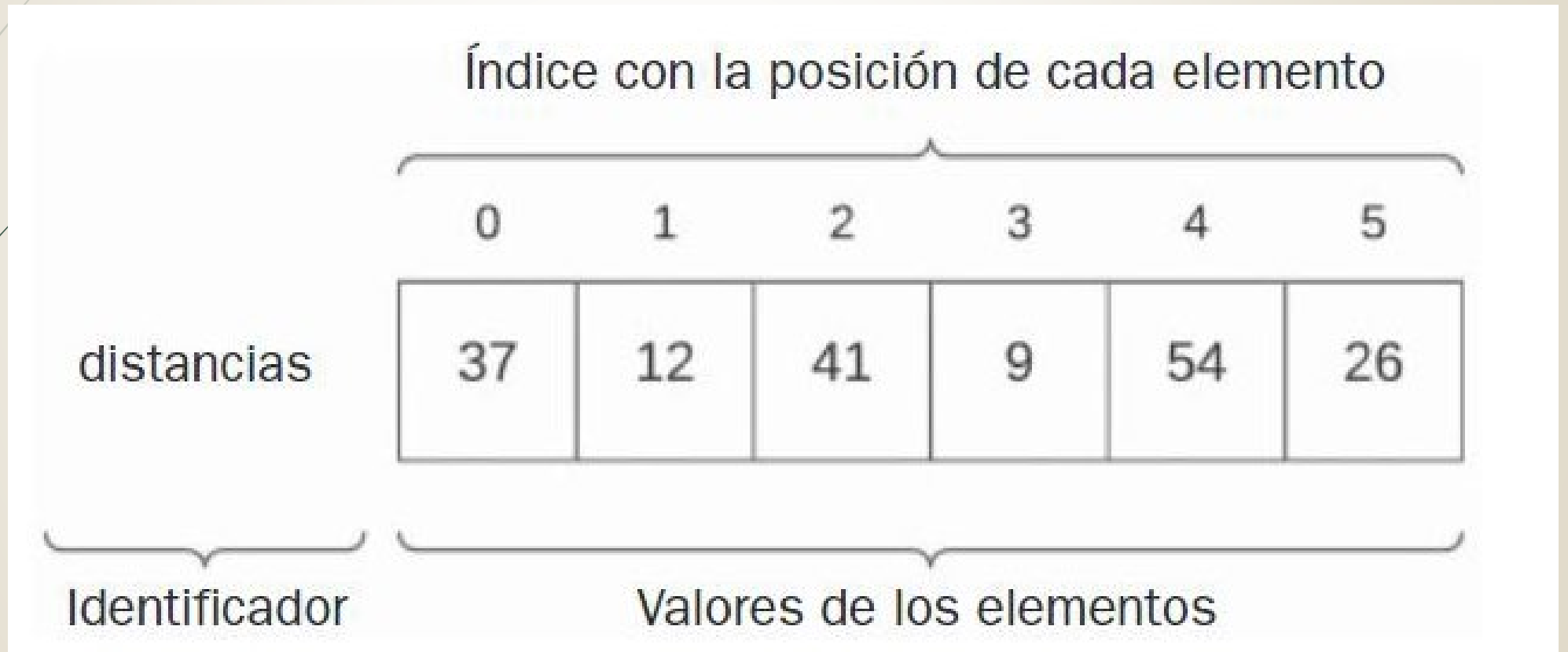
Un array es una estructura de datos (concretamente, un **objeto**) ordenada que permite almacenar múltiples valores bajo un mismo identificador.

## Características:

- Los elementos pueden ser **de cualquier tipo** (number, string, boolean, etc.), incluyendo otros Arrays y objetos.
- Pueden almacenarse **elementos de distinto tipo** dentro del mismo array.
- Su **tamaño es elástico**, es decir, pueden añadirse y eliminarse elementos siempre que sea necesario, y el array se hará más grande o pequeño **de forma dinámica**.
- Cada valor tiene asociado un **índice**.

# Tipos de arrays

- **Unidimensional**

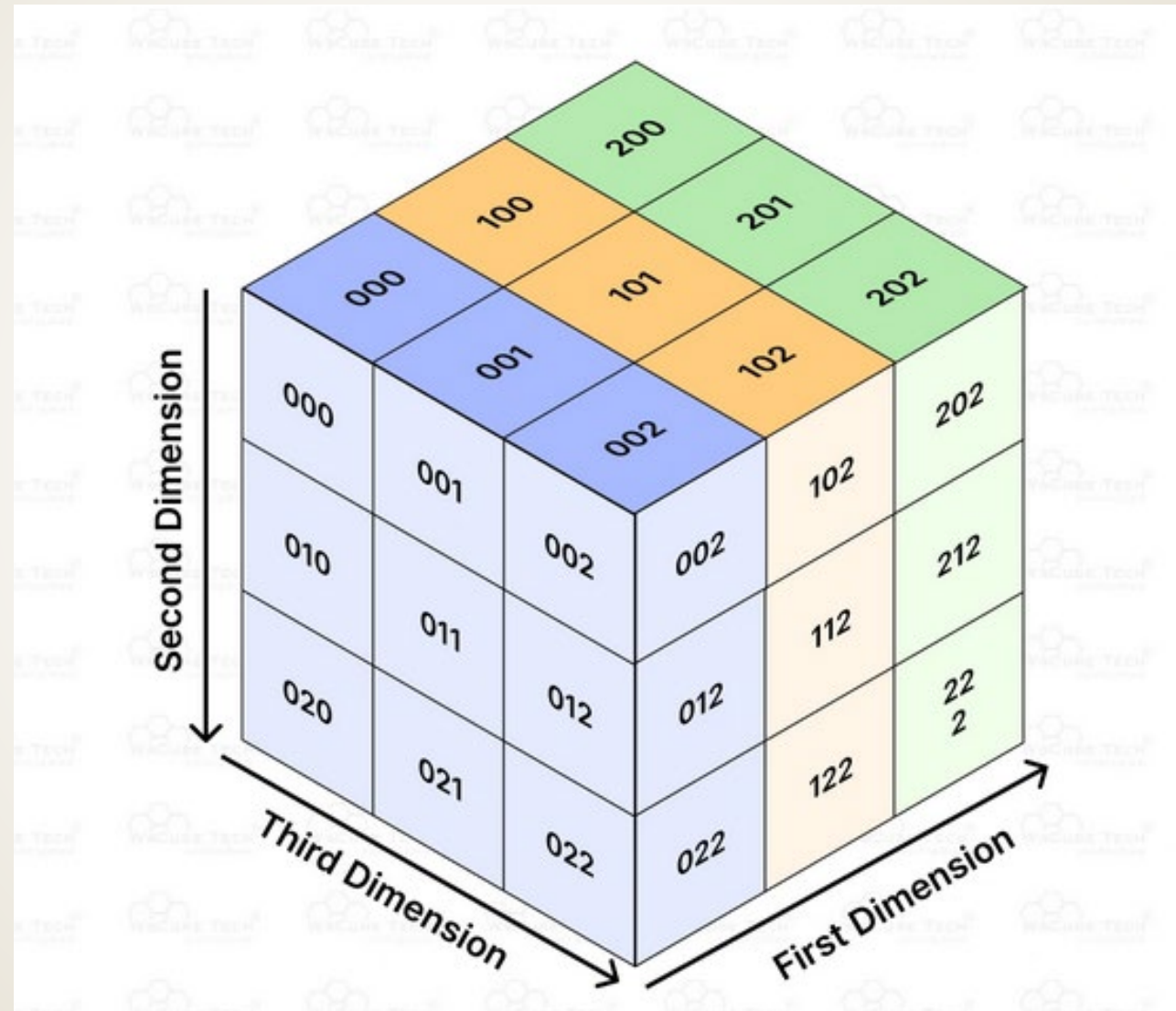


- **Bidimensional**

distancias

		Índice de la columna					
		0	1	2	3	4	5
Índice de la fila	0	37	12	41	9	54	26
	1	11	65	21	7	84	65

## - Tridimensional



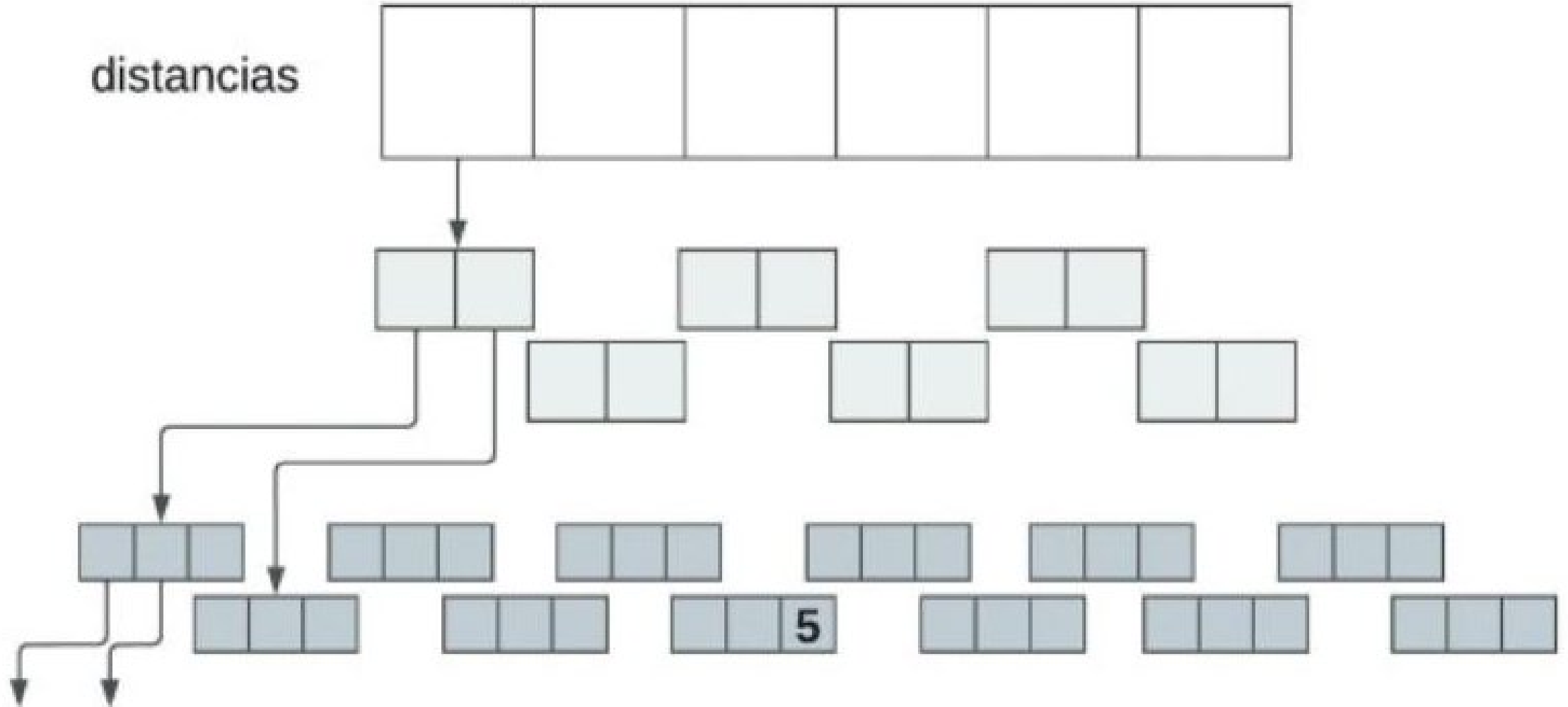
## - **Multidimensional**

Se pueden tener arrays con tantas dimensiones como se desee. La única limitación es la memoria disponible del equipo donde se trabaje.

→ Partimos de un array unidimensional → almacenamos dentro de cada posición otro array unidimensional → dentro de cada una de las posiciones de este segundo array otro array unidimensional → y así indefinidamente hasta completar todas las dimensiones necesarias.



distancias



# Creación y acceso a los elementos

Para crear un array puede utilizarse una de estas tres formas:

```
let array1 = new Array();  
let array2 = Array();  
let array3 = [];
```



.. se crea un array vacío, sin elementos.



```
let array1 = new Array(2);  
let array2 = new Array(3,4);  
let array3 = new Array("Luís");
```

**Igual sin “new”**

- 1ª: crea un array con dos elementos, sin especificar el valor de los elementos que contiene.
- 2ª: crea un array con dos elementos tomando el primer elemento el valor 3 y el segundo elemento el valor 4.
- 3ª: crea un array con un solo elemento cuyo valor es "Luís".

```
let array1 = [2];  
let array2 = [1,3];  
let array3 = ["Luisa"];
```

- 1ª: crea un array con un solo elemento cuyo valor es 2.
- 2ª: crea un array con dos elementos con valores 1 y 3, respectivamente.
- 3ª: crea un array con un solo elemento cuyo valor es "Luisa".

```
let edades = new Array (18,21,34,12,92);  
let edades = Array (18,21,34,12,92);  
let edades = [18,21,34,12,92];
```

Podemos ver su contenido con:

```
console.log(edades);
```

```
▼ (5) [18, 21, 34, 12, 92] ⓘ  
  0: 18  
  1: 21  
  2: 34  
  3: 12  
  4: 92  
  length: 5  
  ► [[Prototype]]: Array(0)
```

**Acceder** al primer elemento y **cambiarle** el valor se puede hacer directamente con:

```
edades[0] = 111;
```

... al tercer elemento:

```
edades[2] = 998;
```

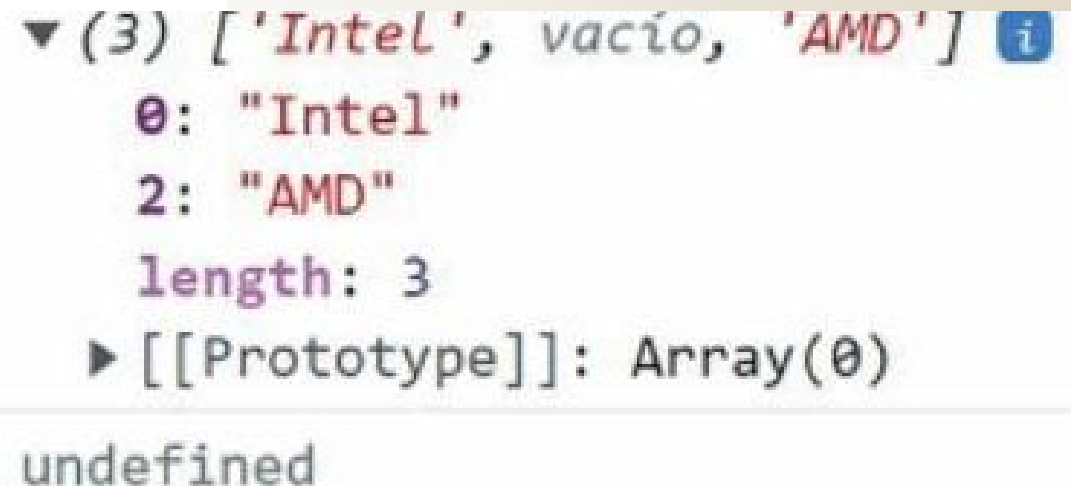
```
▼ (5) [111, 21, 998, 12, 92] ⓘ  
  0: 111  
  1: 21  
  2: 998  
  3: 12  
  4: 92  
  length: 5  
  ► [[Prototype]]: Array(0)
```

Mostrar por consola el contenido del cuarto elemento:

```
console.log(edades[3]);
```

Almacenar en un array tres elementos, pero se desconoce el segundo de ellos:


```
let procesadores = ["Intel",,"AMD"];  
console.log(procesadores);  
console.log(procesadores[1]);
```



```
▼ (3) ['Intel', vacío, 'AMD'] ⓘ  
  0: "Intel"  
  2: "AMD"  
  length: 3  
  ► [[Prototype]]: Array(0)  
undefined
```

```
let procesadores = [,"Intel",,"AMD",];
```

Aquí la última coma se obviaría y no contaría como un nuevo elemento sin definir.



```
▼ (4) [vacío, 'Intel', vacío, 'AMD'] ⓘ  
  1: "Intel"  
  3: "AMD"  
  length: 4  
  ► [[Prototype]]: Array(0)
```



Crear un array bidimensional con dos filas y tres columnas:

```
let tablaNotas = [[,],[,]];
```

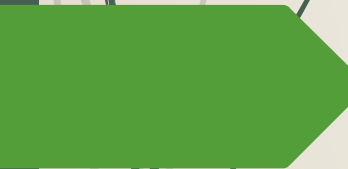
... hay una primera dimensión con dos elementos, y cada uno de ellos definen una segunda dimensión con tres elementos cada una.

```
tablaNotas[0][0] = 1; // Fila 0 - Columna 0  
tablaNotas[0][1] = 2; // Fila 0 - Columna 1  
tablaNotas[0][2] = 3; // Fila 0 - Columna 2  
tablaNotas[1][0] = 4; // Fila 1 - Columna 0  
tablaNotas[1][1] = 5; // Fila 1 - Columna 1  
tablaNotas[1][2] = 6; // Fila 1 - Columna 2
```

Esta forma de crear arrays multidimensionales es bastante engorrosa y nada práctica:

- el uso de las comas puede conducir a errores
- y porque es prácticamente imposible automatizar la creación.

Un array equivalente al anterior sería:



```
let tablaNotas = new Array(2);  
tablaNotas[0] = new Array(3);  
tablaNotas[1] = new Array(3);
```

# Recorrido de un array

## Bucle **for** “clásico”

```
let precios = [69.99,12.49,35.20,99.90];  
for (let i=0; i<precios.length; i++) {  
    console.log(`El precio ${i} es: ${precios[i]}`);  
}
```

→ desventaja: saca todos los elementos del array aunque haya posiciones en los que los elementos no tienen valor, ensuciando la salida.

```
let precios = [69.99,,,12.49,,35.20,,,99.90];
```



El precio 0 es: 69.99

El precio 1 es: undefined

El precio 2 es: undefined

El precio 3 es: 12.49

El precio 4 es: undefined

El precio 5 es: 35.2

El precio 6 es: undefined

El precio 7 es: undefined

El precio 8 es: 99.9

>

## Bucle **for..in**

Opción más limpia y simplificada de recorrer un array indicando la variable que se usará para iterar sobre las posiciones del array.

```
for (let i in precios) {  
    console.log(`El precio ${i} es: ${precios[i]}`);  
}
```

El precio 0 es: 69.99

El precio 3 es: 12.49

El precio 5 es: 35.2

El precio 8 es: 99.9

>

## Bucle **for..of**

Opción más simplificada aún sin necesidad de usar esa variable de iteración.

La iteración se realiza automáticamente y de forma transparente para el usuario. Además, en **cada iteración devuelve el valor de cada elemento del array.**

```
let precios = [69.99,,12.49,,35.20,,,99.90];  
for (let precio of precios) {  
    console.log(precio);  
}
```



→ desventaja: se desconocen los índices de las posiciones, y que en la salida aparecen también los elementos vacíos.


¿Qué bucle **for** se debe usar entonces? Como siempre, la respuesta depende de las necesidades de cada momento.

Recorrer arrays de **cualquier dimensión**: tantos **bucles for anidados** como dimensiones haya.

2 dimensiones

```
let matriz = [[5,1,2],[6,4,3],[9,3,8],[4,1,7]];
```

		índice i		
		0	1	2
índice j	0	5	1	2
	1	6	4	3
	2	9	3	8
	3	4	1	7



```
for (let i=0; i<matriz.length; i++)  
  for (let j=0; j<matriz[i].length; j++) {  
    console.log(`Fila ${i} - Columna ${j}: ${matriz[i][j]}`);  
  }
```

Fila 0 - Columna 0: 5
Fila 0 - Columna 1: 1
Fila 0 - Columna 2: 2
Fila 1 - Columna 0: 6
Fila 1 - Columna 1: 4
Fila 1 - Columna 2: 3
Fila 2 - Columna 0: 9
Fila 2 - Columna 1: 3
Fila 2 - Columna 2: 8
Fila 3 - Columna 0: 4
Fila 3 - Columna 1: 1
Fila 3 - Columna 2: 7
>

De la misma forma para un array de 5 dimensiones:

```
for (let i=0; i<array5d.length; i++)  
  for (let j=0; j<array5d[i].length; j++)  
    for (let k=0; k<array5d[i][j].length; k++)  
      for (let l=0; l<array5d[i][j][k].length; l++)  
        for (let m=0; m<array5d[i][j][k][l].length; m++) {  
          console.log(array5d[i][j][k][l][m]);  
        }  
      }
```

# Manipulación y operaciones con arrays

Vamos a ver algunas de herramientas (**métodos**) que proporciona el propio lenguaje que ahorran una enorme cantidad de tiempo en cada proyecto.

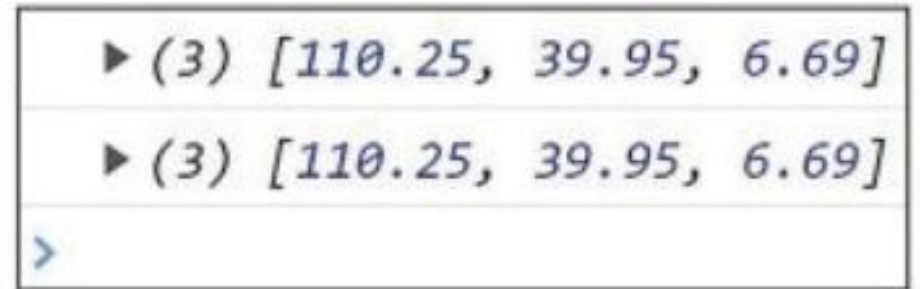
## Asignación de arrays

Partiendo de esta asignación de variables:

```
let unaVariable = 12;  
let otraVariable = unaVariable;  
otraVariable = otraVariable*2;  
console.log(unaVariable); //devuelve 12  
console.log(otraVariable); //devuelve 24
```

Análogamente podríamos intentar hacer esta asignación de arrays:

```
let sinIVA = [20.45,39.95,6.69];  
let conIVA = sinIVA;  
conIVA[0] = 110.25;  
console.log(sinIVA);  
console.log(conIVA);
```



```
► (3) [110.25, 39.95, 6.69]  
► (3) [110.25, 39.95, 6.69]  
>
```

→ El identificador de un array no es donde se almacena la estructura de datos, sino que es una **referencia que apunta a la estructura de datos**. Los dos identificadores apuntan a la misma estructura de datos.

## Adición de elementos a un array

**Directamente**, si sabemos el número de elementos, al final:

```
let elementos = ["a",7,true];  
elementos[3] = 23.45;  
// Resultado ["a",7,true,23.45]
```

Con **push**, si desconocemos el número de elementos, al final:

```
let elementos = ["a",7,true];  
elementos.push("xyz");  
// Resultado ["a",7,true,"xyz"]
```



Con **unshift**, si desconocemos el número de elementos, al principio:

```
let elementos = ["a",7,true];  
elementos.unshift("el primero");  
// Resultado ["el primero","a",7,true,"xyz"]
```

Varios valores en la misma instrucción, con **push** o **unshift**:

```
let elementos = ["a",7,true];  
elementos.unshift("primero","segundo");  
elementos.push("penúltimo","último");  
// Resultado ["primero","segundo","a",7,true,"penúltimo","último"]
```

## Eliminación de elementos de un array

Con **shift** (el primer elemento) o con **pop** (el último elemento), y nos devuelve el elemento eliminado:

```
let elementos = ["a",7,true];  
elementos.shift();  
elementos.pop();  
// Resultado [7]
```

Modificando la propiedad **length**, elimina los elementos que quedan fuera de esa nueva longitud:


```
let elementos = ["a",7,true,90.54,"Lucía",12];  
elementos.length = 2;  
// Resultado ["a",7]
```

Con **splice** eliminamos una cantidad de elementos a partir de una posición, devolviendo los elementos eliminados:

```
let elementos = ["a",7,true,90.54,"Lucía",12];  
let eliminados = elementos.splice(3,2);  
// elementos -> ["a",7,true,12]  
// eliminados -> [90.54,"Lucía"]
```

## Concatenación de arrays

Con **concat** permite extendemos un array añadiéndole al final el contenido de otro array. Ninguno de los dos arrays se modifica, sino que se crea uno nuevo con el contenido de ambos.



```
let original = ["a",7,true,90.54,"Lucía",12];  
let nuevo = [90,80,70];  
let extendido = original.concat(nuevo);  
// original -> ['a', 7, true, 90.54, 'Lucía', 12]  
// nuevo -> [90, 80, 70]  
// extendido -> ['a', 7, true, 90.54, 'Lucía', 12, 90, 80, 70]
```

## Copia de arrays

Con **slice** podemos copiar arrays completos o una parte de ellos.

Si no se indican parámetros, se copia todo el contenido del array en otro array.

Para copiar una parte, el primer parámetro será el índice de la posición desde la que se desea copiar (incluido el elemento que ocupa esa posición), y el segundo el índice de la posición hasta la que se quiere copiar (no incluido el elemento que ocupa esa posición).

```
let original = ["a",7,true,90.54,"Lucía",12];  
let completo = original.slice();  
let parcial = original.slice(2,4);  
// original -> ['a', 7, true, 90.54, 'Lucía', 12]  
// completo -> ['a', 7, true, 90.54, 'Lucía', 12]  
// parcial -> [true, 90.54]
```



## Búsqueda de elementos en un array

**indexOf** devuelve el índice de la posición que ocupa el primer elemento que ha encontrado o -1 si no lo ha encontrado. También permite indicar desde qué posición se desea buscar.

```
let pajar = ["a",7,true,50.54,7,"Marcos"];  
let aguja = 7;  
let resultado = pajar.indexOf(aguja);  
// resultado -> 1
```

```
aguja = 8;  
resultado = pajar.indexOf(aguja);  
// resultado -> -1
```

```
aguja = 7;  
resultado = pajar.indexOf(aguja,2);  
// resultado -> 4
```

**lastIndexOf** hace lo mismo que **indexOf** pero empezando desde el extremo derecho del array.

**includes** nos dice si el elemento existe o no en el array.

```
let pajar = ["a",7,true,50.54,7,"Marcos"];  
let aguja = 7;  
let resultado = pajar.includes(aguja);  
// resultado -> true
```

## Ordenación de arrays

**sort** se utiliza para ordenar los elementos de un array. Este método modifica el array original y devuelve el array ordenado.

```
let vector = [8,4,5,7,1];  
vector.sort();  
// vector -> [1, 4, 5, 7, 8]
```

→ La ordenación de cadenas de caracteres se realiza según el lugar que ocupa cada carácter en la tabla Unicode.

```
let vector = ["Casado","casa","prueba","zancos","ñam"];  
vector.sort();  
// ordenación esperada -> ['casa', 'Casado', 'ñam', 'prueba', 'zancos']  
// ordenación obtenida -> ['Casado', 'casa', 'prueba', 'zancos', 'ñam']
```

→ Habría que establecer criterios propios que modifiquen el comportamiento predeterminado de la ordenación.

**reverse** invierte el orden de los elementos del array

```
let vector = ["Casado","García","Martínez","López","Ballén"];  
vector.sort();  
// ordenación directa-> ['Ballén', 'Casado', 'García', 'López', 'Martínez']  
vector.reverse();  
// ordenación inversa -> ['Martínez', 'López', 'García', 'Casado', 'Ballén']
```

## Conversión entre arrays y cadenas


**toString** todo objeto (y los arrays son un tipo de objeto) tiene definido el método **.toString()** que lo convierte en una cadena con los elementos separados por ','.

```
let miArray = [1, 2, 3];  
let cadenaArray = miArray.toString();  
console.log(cadenaArray); // Muestra "1,2,3"
```



**join** convierte los elementos de un array en una cadena con un separador específico.

```
const frutas = ['Manzana', 'Naranja', 'Banana'];  
const cadena = frutas.join();  
  
console.log(cadena);  
// Resultado: "Manzana,Naranja,Banana"
```



```
const colores = ['rojo', 'verde', 'azul'];  
const cadenaConGuion = colores.join(' - ');  
  
console.log(cadenaConGuion);  
// Resultado: "rojo - verde - azul"
```

split convierte una cadena en un array.

```
let frase = "Hola mundo, ¿cómo estás?";  
let palabras = frase.split(" ");  
// Resultado: ["Hola", "mundo,", "¿cómo", "estás?"]
```

```
let listaNombres = "Ana,Maria,Pedro,Luis";  
let nombres = listaNombres.split(",");  
// Resultado: ["Ana", "Maria", "Pedro", "Luis"]
```

```
let texto = "uno, dos, tres, cuatro, cinco";  
let partes = texto.split(", ", 3);  
// Resultado: ["uno", " dos", " tres"]
```



Más información sobre arrays y sus métodos:

[https://www.w3schools.com/jsref/jsref\\_obj\\_array.asp](https://www.w3schools.com/jsref/jsref_obj_array.asp)

[https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global\\_Objects/Array](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array)

