

TEMA 5.1 – ARRAYS EN JAVASCRIPT

Los **arrays** en JavaScript son una estructura de datos utilizada para almacenar **múltiples valores** en un **mismo identificador**. Los arrays se definen utilizando corchetes `[]` y los elementos se separan por comas. A diferencia de otros lenguajes como C o Java, **no es necesario definir la longitud del array al declararlo**. Además, los arrays pueden contener posiciones vacías (empty), y es posible acceder a sus elementos de manera similar a otros lenguajes utilizando índices.

1.1. Creación y acceso a los elementos:

Para **crear** un array puede utilizarse una de estas tres variantes:

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

En las tres definiciones se crea un array vacío, sin elementos. Pero se debe entender que ocurre en función de lo que se escriba en el interior de los paréntesis:

```
let array1_1 = new Array(3); // Se crea un array con tres posiciones  
let array1_2 = new Array(3,4); // Se crea un array con 2 elementos con valor 3 y 4  
let array1_3 = new Array("Luis"); //Se crea un array con un elemento que tiene valor "Luis"  
  
let array3_1 = [2]; // Se crea un array con un elemento que tiene valor 2  
let array3_2 = [1,3]; //Se crea un array con dos elementos que tienen valores 1 y 3  
let array3_3 = ["Luisa"]; //Se crea un array con un elemento
```

El contenido de un array se puede ver de este modo:

```
console.log(array3_2);
```

```
▼ Array(2) i  
  0: 1  
  1: 3  
  length: 2  
  ► [[Prototype]]: Array(0)
```

[prueba.html:195](#)

Para acceder a un elemento específico se podrá hacer a través de su índice, al igual que se podrá cambiar el valor directamente de esta forma:

```
console.log(array3_2);
console.log(array3_2[0]); //Primer elemento --> 1
array3_2[1] = 2;
console.log(array3_2);
```

▶ (2) [1, 3]	prueba.html:195
1	prueba.html:197
▶ (2) [1, 2]	prueba.html:199

Como ya se ha comentado anteriormente, los arrays pueden almacenar posiciones vacías. Es por ello por lo que podría darse el caso de que, por ejemplo, queramos almacenar en un array tres elementos, pero se desconoce el segundo de ellos:

```
let array4 = ["Intel",, "AMD"];
console.log(array4);
console.log(array4[1]);
```

▶ (3) ['Intel', empty, 'AMD']	prueba.html:201
undefined	prueba.html:202

Esto ocurre si la coma no es el último elemento. Si así fuera, la última coma se obviaría y no contaría como un nuevo elemento sin definir:

```
let array4 = ["Intel",, "AMD",];
console.log(array4);
console.log(array4[3]);
```

▼ (3) ['Intel', empty, 'AMD'] i	prueba.html:201
0: "Intel"	
2: "AMD"	
length: 3	
▶ [[Prototype]]: Array(0)	
undefined	prueba.html:202

1.2. Recorrido de los arrays:

Recorrer un array significa iterar por cada uno de los elementos de este array para realizar alguna operación con los valores que almacena. Para realizar esta tarea se utilizan los bucles **for** y sus variantes. La opción por elegir depende de la necesidad:

- **For loop:**

```
for (let i = 0; i < a.length; i++) {
  console.log(a[i]);
}
```

Esta forma es la más básica y la más utilizada, pero tiene una desventaja y es que saca todos los elementos del array aunque haya posiciones en los que los elementos no tienen valor, ensuciando la salida.

```
let array5 = ["Samsung",, "Balay",, "Cecotec",, "Create"];
for (let i = 0; i < array5.length; i++) {
  console.log(array5[i]);
}
```

Samsung	prueba.html:202
undefined	prueba.html:202
Balay	prueba.html:202
undefined	prueba.html:202
Cecotec	prueba.html:202
undefined	prueba.html:202
Create	prueba.html:202

- **For..in**: Es una opción más simple y a la vez más simplificada de recorrer un array. Basta con indicar la variable que se usará para iterar sobre las posiciones del array y el nombre del propio array.

```
let array5 = ["Samsung",, "Balay",, "Cecotec",, "Create"];
for (let i in array5) {
  console.log("El elemento " + i + " es: " + array5[i]);
}
```

El elemento 0 es: Samsung	prueba.html:208
El elemento 2 es: Balay	prueba.html:208
El elemento 4 es: Cecotec	prueba.html:208
El elemento 6 es: Create	prueba.html:208

Como se puede ver, no es necesario inicializar un contador, ni controlar el tamaño del array ni realizar el incremento del contador. Además no tiene en cuenta los elementos vacíos.

- **For.. of loop**: Esta variante también simplifica bastante el proceso, en este caso no se necesita una variable para cada posición y en cada iteración devuelve el valor de cada elemento del array.

```
for (let i of a) {
  console.log(i);
}
```

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

```
let array5 = ["Samsung", , "Balay", , "Cecotec", , "Create"];
for (let array of array5) {
  console.log(array);
}
```

Samsung	prueba.html:213
undefined	prueba.html:213
Balay	prueba.html:213
undefined	prueba.html:213
Cecotec	prueba.html:213
undefined	prueba.html:213
Create	prueba.html:213

- **ForEach:** es un método que ejecuta una función **callback** en cada elemento de un array. No retorna nada, es solo para realizar alguna acción en cada elemento.

```
let array5 = ["Samsung", , "Balay", , "Cecotec", , "Create"];
array5.forEach((item, index) => {
  console.log("El índice " + index + " tiene el valor: " + item);
});
```

- **Map():** es similar a forEach(), pero con una diferencia importante: **devuelve un nuevo array**. Es útil cuando necesitas transformar los elementos del array.

```
let array5 = ["Samsung", , "Balay", , "Cecotec", , "Create"];

let newArray = array5.map((item, index) => {
  return item.toUpperCase();
});

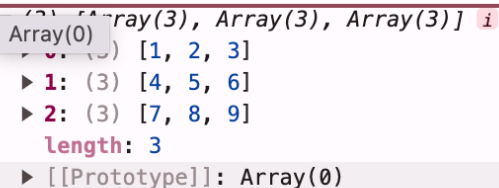
console.log(newArray);
// Output: ['SAMSUNG', 'BALAY', 'CECOTEC', 'CREATE']
```

Hasta ahora, todo lo estudiado sobre la creación y acceso a los elementos de un array se ha realizado con arrays unidimensionales, es decir, aquellos que contienen una sola lista de elementos. Sin embargo, en muchas situaciones, necesitamos organizar datos en más de una

dimensión. Un ejemplo típico es una tabla, que es un array bidimensional, donde hay filas y columnas.

Un **array bidimensional** es simplemente un array que contiene otros arrays como elementos. Cada uno de esos arrays es una fila de la "tabla", y cada elemento de esos arrays es una celda.

```
//Creación de un array bidimensional
let array2D = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
console.log(array2D);
//Acceso a un elemento
console.log(array2D[1][2]); // Segunda fila y tercera columna
```



```
Array(3) Array(3), Array(3), Array(3) i
  0: (3) [1, 2, 3]
  1: (3) [4, 5, 6]
  2: (3) [7, 8, 9]
  length: 3
  [[Prototype]]: Array(0)
```

[prueba.html:222](#)

6

[prueba.html:224](#)

Y para recorrerlo todos los elementos debemos anidar dos bucles for, para las i filas y j para las columnas:

```
//Recorrer el array bidimensional
for (let i = 0; i < array2D.length; i++) {
  for (let j = 0; j < array2D[i].length; j++) {
    console.log("La fila es " + i + " y la Columna " + j + ":" + array2D[i][j]);
  }
}
```

La fila 0 y la Columna 0 es :1	prueba.html:230
La fila 0 y la Columna 1 es :2	prueba.html:230
La fila 0 y la Columna 2 es :3	prueba.html:230
La fila 1 y la Columna 0 es :4	prueba.html:230
La fila 1 y la Columna 1 es :5	prueba.html:230
La fila 1 y la Columna 2 es :6	prueba.html:230
La fila 2 y la Columna 0 es :7	prueba.html:230
La fila 2 y la Columna 1 es :8	prueba.html:230
La fila 2 y la Columna 2 es :9	prueba.html:230

Y así se realizaría sucesivas veces para los diferentes vectores multidimensionales.

1.3. Manipulación y operaciones con arrays:

Los arrays tienen varias propiedades y métodos interesantes que facilitan su manipulación.

Con lo que ya se sabe de arrays tendríamos la capacidad de resolver muchas tareas, pero supondría escribir miles de líneas de código para resolver tareas muy comunes.

- **Añadir elementos a un array:**

- Se puede indicar directamente que guarde un nuevo elemento al final del array.
- **Push():** Agrega uno o más elementos al final del array.
- **Unshift():** Agrega un elemento al principio del array.

```
let array = [1, 2, 3];
array[array.length] = 4; // Agrega 4 al final del array
console.log(array); // Output: [1, 2, 3, 4]

let array = [1, 2, 3];
array.push(4); // Agrega 4 al final
array.push(5, 6); // Agrega varios elementos al final
console.log(array); // Output: [1, 2, 3, 4, 5, 6]

let array = [2, 3, 4];
array.unshift(1); // Agrega 1 al principio
console.log(array); // Output: [1, 2, 3, 4]

array.unshift(0, -1); // Agrega varios elementos al principio
console.log(array); // Output: [-1, 0, 1, 2, 3, 4]
```

► (4) [1, 2, 3, 4]	prueba.html:235
► (6) [1, 2, 3, 4, 5, 6]	prueba.html:240
► (4) [1, 2, 3, 4]	prueba.html:244
► (6) [0, -1, 1, 2, 3, 4]	prueba.html:247

- **Eliminar elementos en un array:**

- **Pop():** Elimina y retorna el último elemento del array. Devuelve el elemento eliminado y en caso de no eliminar nada devuelve undefined.
- **Shift():** Elimina el primer elemento del array. Devuelve el elemento eliminado y en caso de no eliminar nada devuelve undefined.

- También puedo modificar la propiedad **length()** y se elimina todos aquellos elementos que se quedan fuera de la nueva longitud del array.
- También con **splice()** puedo eliminar la cantidad de elementos indicada desde una determinada posición. Devuelve un array con los elementos eliminados.

```
let array9 = [1, 2, 3, 4];
let elementoEliminado9 = array9.pop();
console.log("Array después de pop():", array9); // Output: [1, 2, 3]
console.log("Elemento eliminado con pop():", elementoEliminado9); // Output: 4

let array10 = [1, 2, 3, 4];
let primerElementoEliminado10 = array10.shift();
console.log("Array después de shift():", array10); // Output: [2, 3, 4]
console.log("Elemento eliminado con shift():", primerElementoEliminado10); // Output: 1

let array11 = [1, 2, 3, 4, 5];
array11.length = 3;
console.log("Array después de modificar length:", array11); // Output: [1, 2, 3]

let array12 = [1, 2, 3, 4, 5];
let elementosEliminados12 = array12.splice(1, 2);
console.log("Array después de splice():", array12); // Output: [1, 4, 5]
console.log("Elementos eliminados con splice():", elementosEliminados12); // Output: [2, 3]
```

Array después de pop(): ▶ (3) [1, 2, 3]	prueba.html:253
Elemento eliminado con pop(): 4	prueba.html:254
Array después de shift(): ▶ (3) [2, 3, 4]	prueba.html:259
Elemento eliminado con shift(): 1	prueba.html:260
Array después de modificar length: ▶ (3) [1, 2, 3]	prueba.html:265
Array después de splice(): ▶ (3) [1, 4, 5]	prueba.html:270
Elementos eliminados con splice(): ▶ (2) [2, 3]	prueba.html:271

- **Length:** Devuelve el número de elementos en el array.
- **Sort():** Ordena los elementos del array. Esta operación muta el array original, pero si no queremos mutarlo podemos usar toSorted().

```
console.log([2,3,44,5,"1","a"].sort());
```

▶ (6) ['1', 2, 3, 44, 5, 'a'] [prueba.html:274](#)

Se debe tener en cuenta que la ordenación se realiza teniendo en cuenta el lugar que ocupa cada carácter en la tabla Unicode. Por ello al realizar esta ordenación ocurre un resultado inesperado:

```
let array16 = ["Casado", "casa", "prueba", "ñam", "zancos"];
array16.sort();
console.log(array16);
```

► (5) ['Casado', 'casa', 'prueba', 'zancos', 'ñam'] [prueba.html:334](#)

La ordenación que esperamos era: “casa”, “Casado”, “ñam”, “prueba” y “zancos”.

Esto ocurre porque en **Unicode** aparece antes la C que la c. Igualmente ocurre con la ñ que aparece mucho después que el resto de las letras.

Para solventar este contratiempo y modificar el comportamiento predeterminado de la ordenación, podemos usar **criterios de ordenación personalizados** en JS pero para ello habrá que pasarle una **función de comparación**. Esta función de comparación debe retornar un número:

- Valor negativo si el primer argumento debe aparecer antes que el segundo.

```
let array18 = [10, 5, 20, 15];

// Ordenar de forma descendente
array18.sort((a, b) => b - a);
console.log("Descendente:", array18); // Output: [20, 15, 10, 5]
```

- Un valor positivo si el primer argumento debe aparecer después del segundo.

```
let array17 = [10, 5, 20, 15];

// Ordenar de forma ascendente
array17.sort((a, b) => a - b);
console.log("Ascendente:", array17); // Output: [5, 10, 15, 20]
```

- Reverse() que resuelve la necesidad de darle la vuelta a un array:

```
let palabras = ['manzana', 'pera', 'plátano', 'naranja'];

palabras.sort();
console.log("Ascendente:", palabras); // Output: ['manzana', 'naranja', 'pera', 'plátano']

palabras.reverse();
console.log("Descendente:", palabras); // Output: ['plátano', 'pera', 'naranja', 'manzana']
```


- **Concat():** Permite extender 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 array13 = [1, 2, 3];
let array14 = [4, 5, 6];

let nuevoArray = array13.concat(array14);

console.log("Array original 1:", array13); // Output: [1, 2, 3]
console.log("Array original 2:", array14); // Output: [4, 5, 6]
console.log("Nuevo array combinado:", nuevoArray); // Output: [1, 2, 3, 4, 5, 6]
```

Array original 1: ▶ (3) [1, 2, 3]	prueba.html:289
Array original 2: ▶ (3) [4, 5, 6]	prueba.html:290
Nuevo array combinado: ▶ (6) [1, 2, 3, 4, 5, 6]	prueba.html:291

- **Slice():** Los arrays se pueden copiar completos o traer una parte de ellos usando slice. Si no se indican parámetros, se copia todo el contenido del array en otro array. Si lo que se quiere hacer es copiar solo una parte, hay que indicar dos parámetros:
 - El primero será el índice de la posición desde la que se desea copiar (incluido el elemento que ocupa)
 - El segundo el índice de la posición hasta la que se quiere copiar (no incluido el elemento)

```
let array15 = [10, 20, 30, 40, 50, 60, 70, 80];

let copiaCompleta = array15.slice();
console.log("Copia completa del array:", copiaCompleta);

let copiaParcial = array15.slice(2, 5);
console.log("Copia parcial del array:", copiaParcial);
```



- **Para la búsqueda de elementos en un array:**
 - **Find():** Retorna el primer elemento que cumple con la condición dada.
 - **FindIndex():** Devuelve el índice del primer elemento que cumple con la condición dada.
 - **IndexOf():** Devuelve el primer índice en el que se puede encontrar un elemento o -1 si no lo ha encontrado. También permite indicar desde que posición se desea buscar. **LastIndex()** realiza la misma tarea que **indexOf()** pero trabaja desde el extremo derecho del array.
 - **Includes():** Comprueba si un array contiene un determinado elemento.
 - **Filter():** Crea un nuevo array con todos los elementos que cumplen la condición.

```
let array15 = [10, 20, 30, 40, 50, 20, 60];

let primerMayorDe25 = array15.find(elemento => elemento > 25);
console.log("Primer elemento mayor a 25:", primerMayorDe25); // Output: 30

let indiceMayorDe25 = array15.findIndex(elemento => elemento > 25);
console.log("Índice del primer elemento mayor a 25:", indiceMayorDe25); // Output: 2

let indice20 = array15.indexOf(20);
console.log("Índice del primer 20:", indice20); // Output: 1

let ultimoIndice20 = array15.lastIndexOf(20);
console.log("Último índice del 20:", ultimoIndice20); // Output: 5

let contiene40 = array15.includes(40);
console.log("El array contiene 40:", contiene40); // Output: true

let mayoresDe30 = array15.filter(elemento => elemento > 30);
console.log("Elementos mayores a 30:", mayoresDe30); // Output: [40, 50, 60]
```

- **Join():** Transforma un array en una cadena.

```
let arrayNumeros = [1, 2, 3];
let cadena = arrayNumeros.join('-');
console.log(cadena); // Output: "1-2-3"
```

- **Split():** Transforma una cadena en un array.

```
let cadenaTexto = "a,b,c,d";
let arrayPartes = cadenaTexto.split(',');
console.log(arrayPartes); // Output: ["a", "b", "c", "d"]
```

- **Fill():** Llena un array con un valor específico.

```
let arrayVacio = new Array(5).fill(0);
console.log(arrayVacio); // Output: [0, 0, 0, 0, 0]
```

- **Reduce():** toma un array y lo reduce a **un único valor**. Lo hace ejecutando una función sobre cada elemento del array, acumulando los resultados en un solo valor.

```
let numerosArray = [1, 2, 3];

let suma = numerosArray.reduce((acumulador, actual) => acumulador + actual, 0);

console.log(suma); // Output: 6
```

- **Some():** Verifica si **al menos uno** de los elementos en un array cumple con una condición. Devuelve true si algún elemento pasa la prueba, o false si ninguno lo hace.

```
let numeros = [3, 7, 9, 12, 4];
let condicion = numeros.some(numero => numero > 10);

console.log(condicion)
```

EJERCICIOS DE ARRAYS:

1. Crea un array llamado números con 10 números aleatorios entre 1 y 100. Crea una función que reciba ese array y devuelva el valor mínimo y el valor máximo.

2. Crea un array llamado edades con 10 números. Filtra las edades mayores de 18, luego suma 5 años a cada una de las edades filtradas. Convierte el resultado en una cadena de texto separada por comas.
3. Dado un array llamado nombres con 6 nombres de personas, ordénalo alfabéticamente. Verifica si el nombre "Pedro" está presente en el array. Encuentra el primer nombre que tenga más de 5 letras. Crea un nuevo array que contenga solo los primeros 3 nombres.
4. Crea un array llamado ciudades con 5 ciudades. Agrega dos ciudades más al final del array y elimina la última ciudad agregada. Elimina la segunda ciudad del array y reemplázala por otra ciudad. Invierte el orden de las ciudades.
5. Crea una función que calcule la media de un array de números. Usa la función con un array de 6 números generados aleatoriamente entre 0 y 50. Ordena los números antes de calcular la media.
6. Crea un array llamado frutas con 5 nombres de frutas. Convierte todas las frutas a mayúsculas, luego obtén un nuevo array con las frutas cuyos nombres tengan más de 6 letras. Únelas con otro array de dos frutas nuevas.
7. Dado un array de números llamado ventas con los valores [100, 200, 150, 400, 300], calcula la suma total de todas las ventas. Luego agrega un 10% de impuesto a cada venta. Verifica si alguna venta es mayor de 500.