Unidad 3
Estructuras de datos

CESUR
Tu Centro Oficial de FP

1. Introducción

Vamos a estudiar los tipos de datos complejos, es decir, estructuras de datos organizados en colecciones que pueden manipularse como si fueran una única unidad.

Estos tipos de datos, a su vez, se pueden combinar con otros para crear estructuras más complejas.

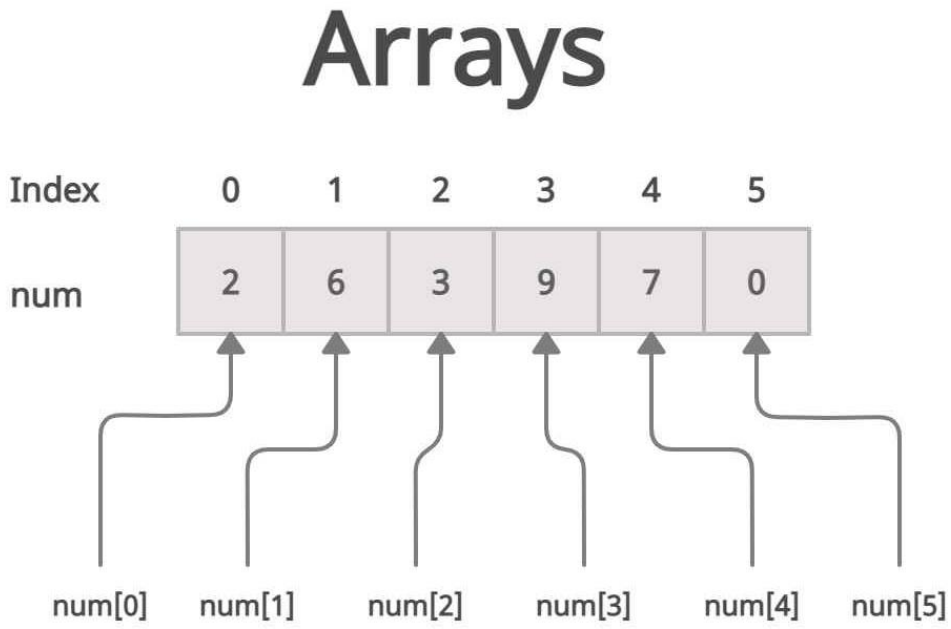
2. Arrays

ARRAYS – estructuras de datos ordenada que permite almacenar múltiples valores bajo un mismo identificador.

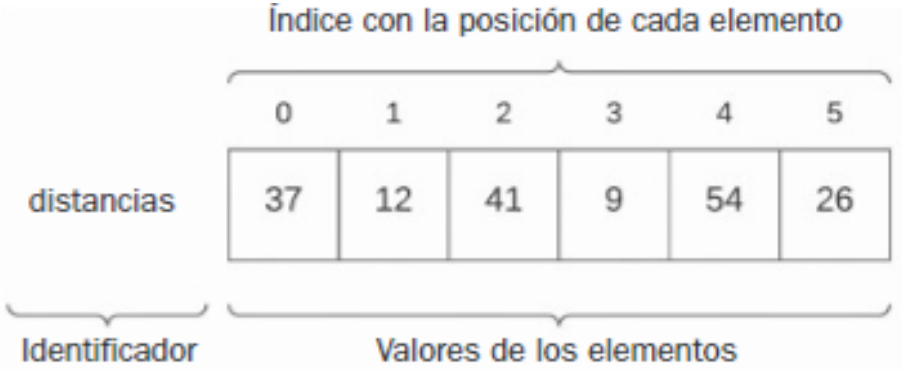
JS permite almacenar elementos de distinto tipo dentro del mismo array.

Además, su tamaño es elástico, es decir, pueden añadirse y eliminarse elementos siempre que sea necesario, de tal forma que el tamaño del array es dinámico.

Los arrays tienen un nombre que los identifica y contienen una serie de valores. A cada valor, se le asocia un **índice**.



2. Arrays



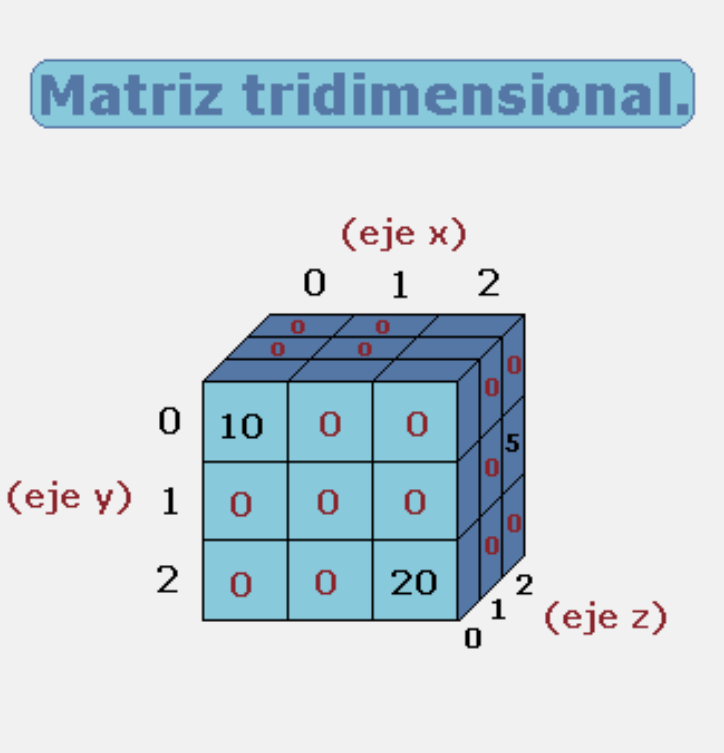
ARRAY UNIDIMENSIONAL (distancias) – tiene 6 elementos numéricos. Para acceder a cada elemento, se hace referencia al índice de la posición que ocupa en el array.

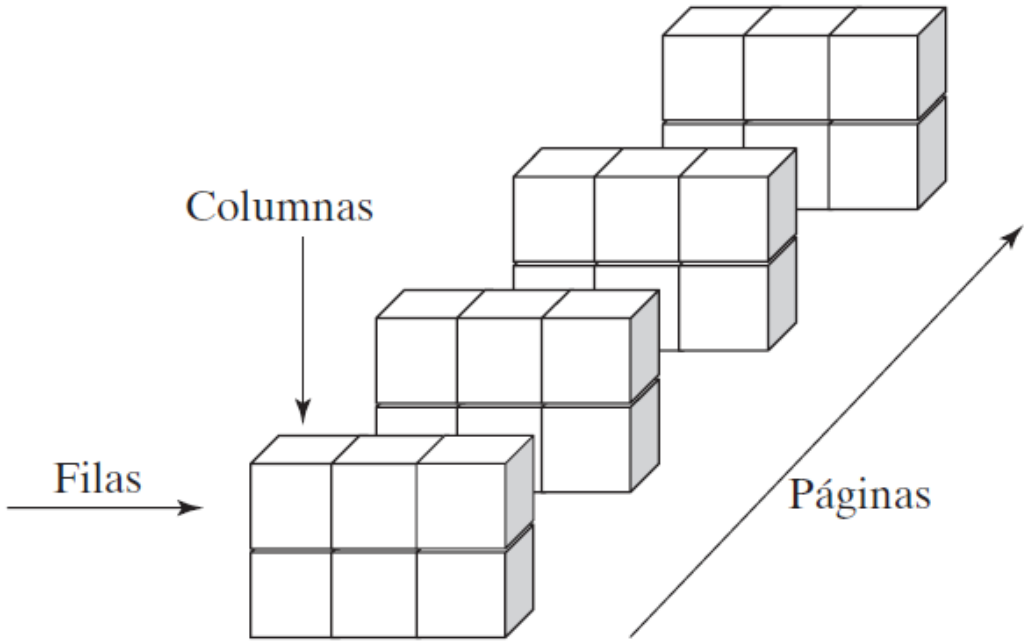
ARRAY BIDIMENSIONAL – en este caso, el array almacena 6 elementos por cada una de sus filas, es decir, en total 12 elementos numéricos.

Para acceder a los elementos del array hay que indicar la fila y columna en la que se sitúan.

Por ejemplo, para acceder al elemento que vale 7, hay que indicarle que está en la fila 1 y en la columna 3.

ARRAY TRIDIMENSIONAL – se utiliza una representación espacial compuesta por 3 ejes (X, Y, Z).

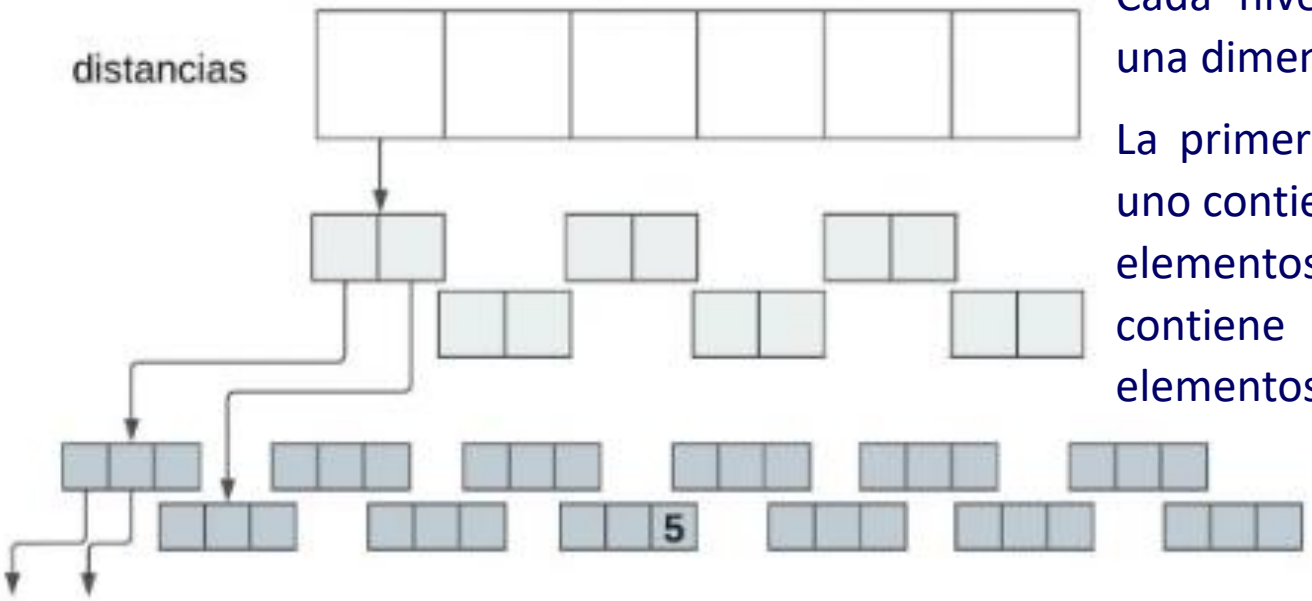




2. Arrays

ARRAY MULTIDIMENSIONAL – se pueden crear arrays de más de 3 dimensiones, de hecho, se pueden crear arrays con tantas dimensiones como se desee, la única limitación es la memoria disponible del equipo donde se trabaje.

Para imaginarse un array multidimensional, partimos de un array unidimensional y almacenando de dentro de casa posición otro array unidimensional y dentro de cada una de las posiciones de este segundo array, otro array unidimensional y así indefinidamente hasta completar todas las dimensiones necesarias.



Cada nivel de anidamiento supone una dimensión.

La primera tiene 6 elementos, cada uno contiene a su vez un array de dos elementos y cada uno de ellos contiene a su vez un array de 3 elementos.

Para acceder a un valor, basta con indicar el índice de cada dimensión.

CREACIÓN Y ACCESO A LOS ELEMENTOS DE UN ARRAY

Para crear un array podemos hacerlo de 3 formas:

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

Estas tres formas, crean un array vacío, sin elementos.

Si queremos crear un array con elementos, podemos hacerlo de la siguiente forma:

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

array1 – crea un array con dos elementos, sin especificar el valor de los elementos que contiene.

array2 – crear un array con dos elementos, tomando el primer elemento el valor 3 y el segundo elemento el valor 4

array3 – crear un array con un único elemento cuyo valor es “Luis”.

NOTA: Si utilizamos **Array()** sólo, en vez de **new Array()** los resultados serían los mismos.

2. Arrays

Si usamos corchetes, sucede lo siguiente:

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

array1 – crea un array con un elemento cuyo valor es 2.

array2 – crear un array con dos elementos, tomando el primer elemento el valor 1 y el segundo elemento el valor 3

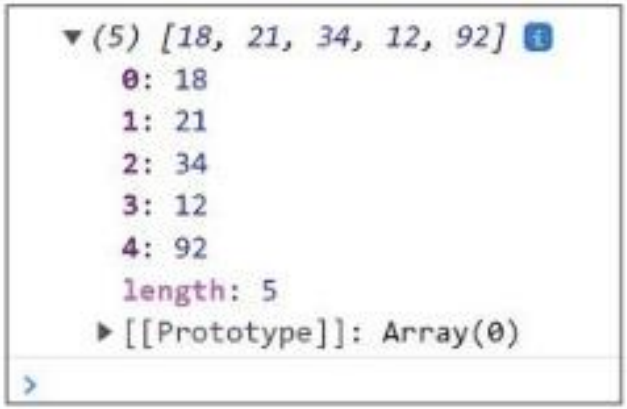
array3 – crear un array con un único elemento cuyo valor es “Luisa”.

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

El array edades se puede crear de estas 3 formas y el resultado es equivalente. Crea un array de 5 elementos, cuyos valores son: 18, 21, 34, 12 y 92.

Para ver el contenido del array utilizamos:
console.log(edades);

En la consola del navegador se mostrará su estructura, contenido, longitud (número de elementos del array y qué posición ocupa cada elemento.



2. Arrays

Si queremos cambiar el valor de un elemento del array, por ejemplo, el primero y el tercero, lo haríamos:

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

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

El array edades quedaría:

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

Importante

A los elementos de un *array* se accede contando las posiciones desde el 0. Si un *array* tiene cinco elementos, se accede a ellos utilizando los índices 0, 1, 2, 3 y 4.

Cuando se dice «acceder al tercer elemento», se está accediendo al elemento cuyo índice es 2.

Es muy importante interiorizar esto porque suele ser una fuente interminable de errores, incluso entre los programadores más experimentados.

2. Arrays

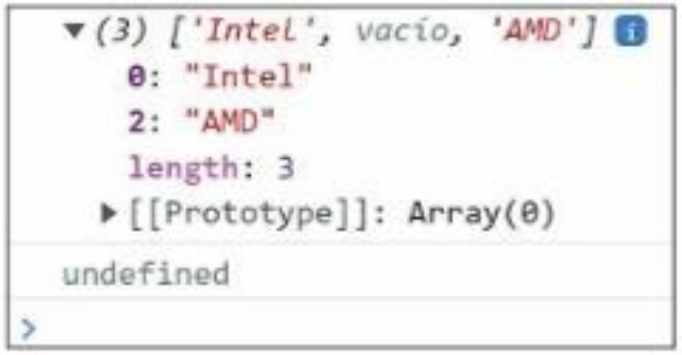
Si queremos mostrar un elemento de un array, sólo hay que pensar en la posición que ocupa dentro del array, se localiza su índice y se trata la expresión como si fuera una variable más.

Por ejemplo, para mostrar por consola el contenido del cuarto elemento pondremos:

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

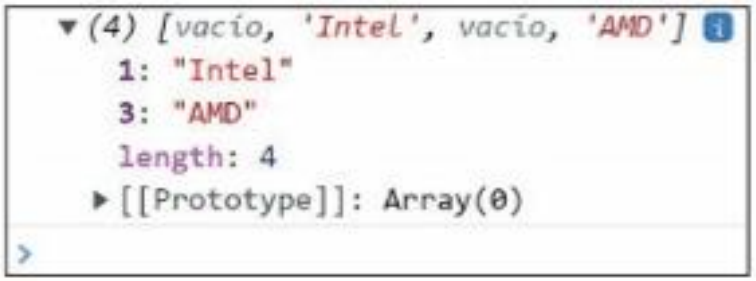
El uso de comas que separan los elementos durante la creación de arrays es muy importante:

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



Esta expresión genera un valor undefined en la segunda posición

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



Esta expresión genera cuatro elementos, estando el primero y el tercero sin definir.

Ejemplo 1: ARRAYS

Crea una página web, que incluya un array que almacene el nombre de cinco localidades. Debe mostrar por pantalla:

1) Las cinco localidades
2) Las localidades que ocupen las posiciones pares
3) Las localidades que ocupen las posiciones pares

NOTA: Para saber la longitud de un array se utiliza el objeto **nombreArray.length**

Ejemplo 1: ARRAYS UNIDIMENSIONALES

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <meta charset="UTF-8">
5           <title>Actividad 1</title>
6           <script>
7               let localidades = ["Sanlúcar", "Chipiona","Rota","Barbate","Tarifa"];
8               document.write(localidades);
9               document.write("<br>");
10              let i=0;
11              while (i<localidades.length){
12                  if (i % 2){
13                      document.write(i+" - " + localidades[i]+", ");
14                  }
15                  i++;
16              }
17              document.write("<br>");
18              let j=0;
19              while (j<=localidades.length){
20                  if (!(j % 2)){
21                      document.write(j+" - " +localidades[j]+", ");
22                  }
23                  j++;
24              }
25          </script>
26      </head>
27      <body>
28
29      </body>
30  </html>
```

Sanlúcar,Chipiona,Rota,Barbate,Tarifa
1 - Chipiona, 3 - Barbate,
0 - Sanlúcar, 2 - Rota, 4 - Tarifa,

2. Arrays

Para crear un array bidimensional de 2 filas y 3 columnas, lo hacemos:

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

Para almacenar valores en dicho array hay que almacenar un valor por cada posición del array:

```
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 bidimensionales es muy engorrosa y nada práctica porque el uso de comas puede conducir a errores y segundo porque es prácticamente imposible automatizar la creación.

2. Arrays

Para crear arrays bidimensionales es más fácil utilizar la palabra reservada new y creando el array con sucesivas llamadas a esa instrucción.

El anterior array se declararía:

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

Para rellenar los valores se haría de la misma forma que vimos anteriormente:

```
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
```

Ejemplo 2: ARRAY

Crea una página web, que incluya un array que asigne a cada color RGB un alias y muestre por consola la información poniendo cada texto en su color correspondiente:

- Naranja: #F39C12
- Lima: #C0F312
- Turquesa: #12F3E5
- Rosa: #F312AF
- Rojo: #F31212

Ejemplo 2: ARRAY

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <meta charset="UTF-8">
5           <title>Actividad 1</title>
6           <script>
7               //Array con nombres de colores
8               let nombresColores = ["Naranja", "Lima","Turquesa","Rosa","Rojo"];
9
10              //Array de valores RGB de cada color
11              let valoresColores = ["#F39C12", "#C0F312", "#12F3E5", "#F312AF", "#F31212"];
12
13              // Mostrar los nombre y valores de los colores
14              for (let i=0; i<nombresColores.length; i++){
15                  console.log('%c'+ nombresColores[i] + ': ' + valoresColores[i],("color:" + valoresColores[i]+";"));
16              }
17          </script>
18      </head>
19      <body>
20
21      </body>
22  </html>
```

```
Naranja: #F39C12                    UD3_Ejemplo2_ARRAYS_…MENSIONALES.html:15
Lima: #C0F312                       UD3_Ejemplo2_ARRAYS_…MENSIONALES.html:15
Turquesa: #12F3E5                   UD3_Ejemplo2_ARRAYS_…MENSIONALES.html:15
Rosa: #F312AF                       UD3_Ejemplo2_ARRAYS_…MENSIONALES.html:15
Rojo: #F31212                       UD3_Ejemplo2_ARRAYS_…MENSIONALES.html:15
```

RECORRIDO DE UN ARRAY

Recorrer un array significa acceder a cada uno de los elementos que contine para realizar alguna operación con los valores que almacena.

MÉTODO 1: BUCLE FOR

Para recorrer los arrays podemos utilizar bucles **for**.

```
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]}`);
}
```

En este fragmento de código se define el *array* **precios** con cuatro elementos. Después se realiza un recorrido del *array* utilizando una nueva variable, **i**, para iterar sobre sus elementos tomando como límite del *array* la cantidad de sus elementos (**.length** devuelve el número de elementos del *array*)

```
El precio 0 es: 69.99
El precio 1 es: 12.49
El precio 2 es: 35.2
El precio 3 es: 99.9
>
```

Este bucle tiene la desventaja de que saca todos los elementos del array aunque haya posiciones sin valor.

```
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]}`);
}
```

```
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
>
```

MÉTODO 2: BUCLE FOR..IN

Existe otra forma más simplificada de recorrer un array con el bucle **for..in**.

Sólo hay que indicar la variable que se va a utilizar para recorrer el array y el nombre del propio array.

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

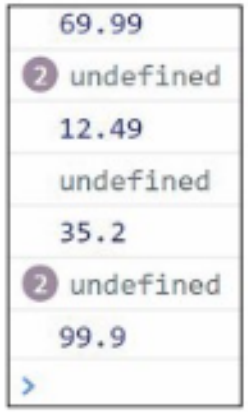
```
El precio 0 es: 69.99
El precio 3 es: 12.49
El precio 5 es: 35.2
El precio 8 es: 99.9
>
```

Como se puede ver, no es necesario inicializar el contador, ni controlar el tamaño del *array*, ni tampoco realizar el incremento del contador. Su funcionamiento es automático, y además no tiene en cuenta los elementos vacíos

MÉTODO 3: BUCLE FOR..OF

Este método simplifica aún más el proceso. No se utiliza una variable para recorrer el array, sino que se realiza automáticamente y de forma transparente para el usuario.

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



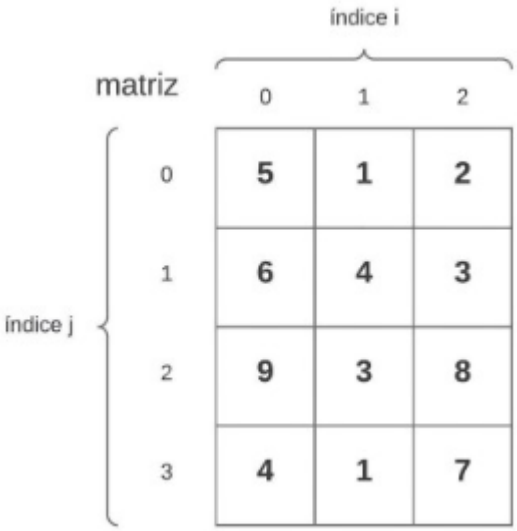
Su desventaja es que se desconocen los índices de las posiciones, y que en la salida aparecen también los elementos vacíos

MÉTODO 4: BUCLE FOREACH

Este método se estudiará con las funciones.

2. Arrays

Para recorrer arrays de cualquier dimensión, lo que hacemos es anidad un bucle for dentro de otro. Anidaremos tantos bucles for como dimensiones tenga el array.



Como se había adelantado, estamos ante un *array* de dos dimensiones, por tanto, en la solución deben aparecer dos bucles **for** anidados. Se utiliza un contador para recorrer los índices de cada dimensión (**i** para las filas y **j** para las columnas), teniendo la precaución de modificar la expresión que controla el número máximo de elementos del *array* (**matriz [i].length** para el caso de las filas, y **matriz[j].length** para el caso de las columnas).

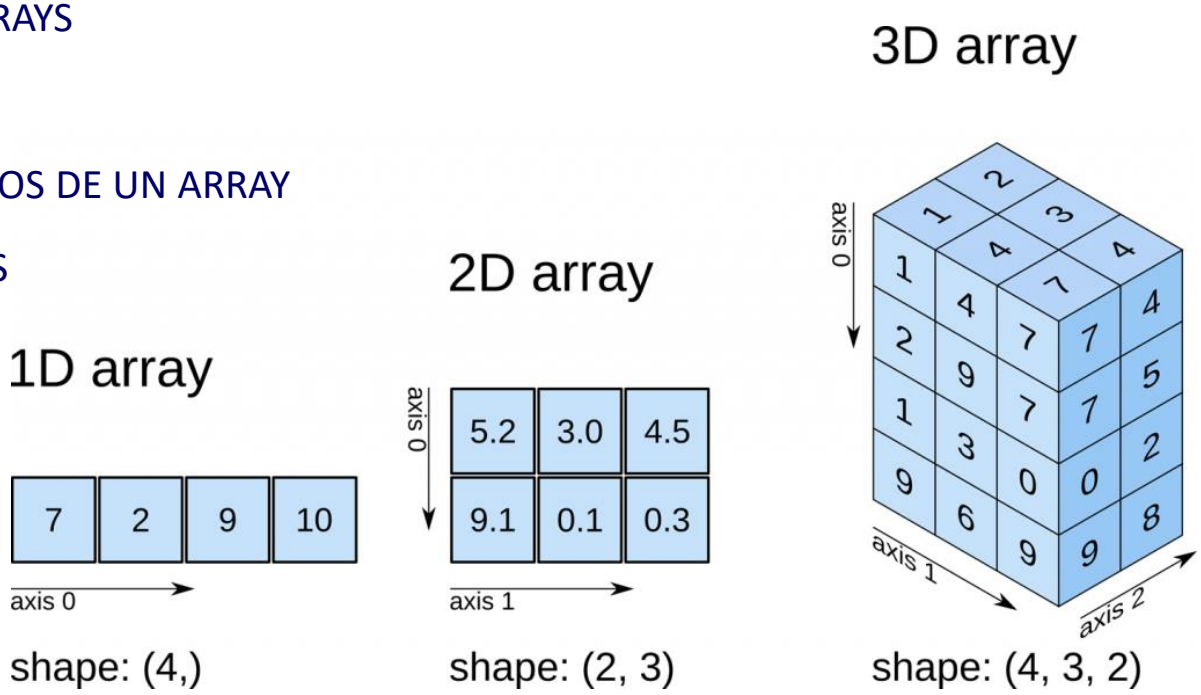
```
let matriz = [[5,1,2],[6,4,3],[9,3,8],[4,1,7]];

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
>
```

3. Manipulación y operaciones con arrays

JS proporciona ciertas herramientas para trabajar con arrays:

1) ASIGNACIÓN DE ARRAYS
2) ADICCIÓN DE ELEMENTOS A UN ARRAY
3) ELIMINACIÓN DE ELEMENTOS DE UN ARRAY
4) CONCATENACIÓN DE ARRAYS
5) COPIA DE ARRAYS
6) BÚSQUEDA DE ELEMENTOS DE UN ARRAY
7) ORDENACIÓN DE ARRAYS



3. Manipulación y operaciones con arrays

1) ASIGNACIÓN DE ARRAYS

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.

```
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]
>
```

Cuando se asigna identificadores a los arrays, lo que ocurre es que dos identificadores apuntan a la misma estructura de datos, por lo que cualquier modificación que se haga utilizando cualquiera de los dos identificadores estará afectando a la misma estructura de datos.

3. Manipulación y operaciones con arrays

2) ADICCIÓN DE ELEMENTOS A UN ARRAY

Podemos hacerlo de diferentes formas:

1. Si se sabe cuántos elementos tiene un array se le puede indicar directamente que guarde un nuevo elemento al final

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

2. Si se desconoce el número de elementos se puede utilizar **push**, que añade directamente un elemento al final (y además devuelve el número de elementos del array tras la inserción).

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

3. Se puede añadir un elemento al principio con **unshift**.

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

NOTA: tanto **unshift** como **push** permiten añadir en una misma instrucción varios valores, los que necesitemos.

3. Manipulación y operaciones con arrays

3) ELIMINACIÓN DE ELEMENTOS DE UN ARRAY

Podemos hacerlo de diferentes formas:

1. **Utilizando los homólogos a unshift y push:**
   a. **shift** – elimina el primer elemento
   b. **pop** – elimina el último elemento

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

En ambos casos se devuelve el elemento eliminado, y en caso de no haber elementos, se devuelve **undefined**.

2. **Modificar la propiedad length**

Se eliminan todos aquellos elementos que se quedan fuera de la nueva longitud del array:

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

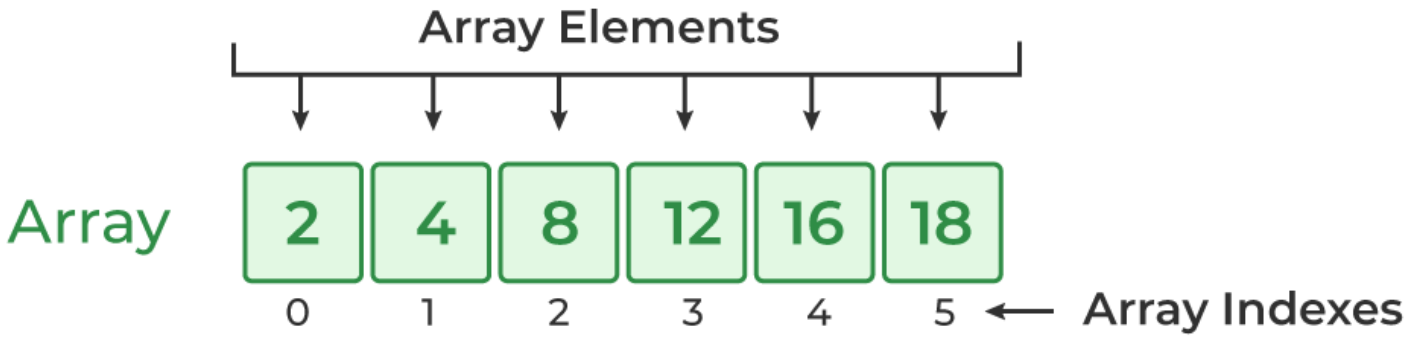
3. Manipulación y operaciones con arrays

3. Eliminar la cantidad de elementos indicada desde una posición determinada con splice

Además, devuelve un array con 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"]
```

Este código elimina los dos primeros elementos que encuentra desde la posición 3, es decir, elimina los elementos de las posiciones 4 y 5.



3. Manipulación y operaciones con arrays

Ejemplo 3: SPLICE

Crear un array que incluya los números del 0 al 9.

Utilizando el método **splice** elimina 3 componentes a partir de la segunda posición del array.

Mostrar cómo era el array inicialmente y como queda después de la utilización de splice.

```
Array inicial
0
1
2
3
4
5
6
7
8
9
Array después de utilizar splice
0
4
5
6
7
8
9
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ejemplo de JavaScript</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <script>
            let vec = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
            document.write('Array inicial<br>');
            for (let f = 0; f < vec.length; f++) {
                document.write(vec[f] + '<br>');
            }
            vec.splice(1, 3);
            document.write('Array después de utilizar splice<br>');
            for (let f = 0; f < vec.length; f++) {
                document.write(vec[f] + '<br>');
            }
        </script>
    </body>
</html>
```

3. Manipulación y operaciones con arrays

4) CONCATENACIÓN DE ARRAYS

Concat permite extender un array añadiéndole al final el contenido de otro array.

Ninguno de los arrays se modifica, sino que se crea uno nuevo con el contenido de ambos.

El array cuyos elementos aparecerán en primer lugar será aquel que haga uso de **concat**:

```
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]
```

3. Manipulación y operaciones con arrays

Ejemplo 4: CONCAT

Crear dos arrays con 4 elementos. Mostrar en la página:

1) Los datos del primer array
2) Los datos del segundo array
3) Los datos de los dos arrays anteriores concatenados

```
Primer vector:
10,20,30,40

Segundo vector:
100,200,300,400

Vectores concatenados:
10,20,30,40,100,200,300,400
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ejemplo de JavaScript</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <script>
            let vec1 = [10, 20, 30, 40];
            let vec2 = [100, 200, 300, 400];
            let vecsuma = vec1.concat(vec2);
            document.write('Primer vector:</br>');
            document.write(vec1.join() + '<br></br>');
            document.write('Segundo vector:</br>');
            document.write(vec2.join() + '<br></br>');
            document.write('Vectores concatenados:</br>');
            document.write(vecsuma.join() + '<br>');
        </script>
    </body>
</html>
```

3. Manipulación y operaciones con arrays

5) COPIA DE ARRAYS

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 esa posición), y el segundo el índice de la posición hasta la que se quiere copiar (no incluido el elemento que ocupa esa posición).

El **método slice**, no modifica el vector original, sino que retoma otro vector con las componentes indicadas.

```
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]
```

3. Manipulación y operaciones con arrays

Ejemplo 5: SLICE

Crear un array con 6 elementos.

Mostrar en la página:

1) El array original
2) Un array que muestre desde la posición 1 hasta la posición 4 sin incluirla.

```
Array original
10-20-30-40-50-60-
Array resultante de la llamada a slice
20-30-40-
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ejemplo de JavaScript</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <script>
            let vec = [10, 20, 30, 40, 50, 60];
            document.write('Array original<br>');
            for (let f = 0; f < vec.length; f++) {
                document.write(vec[f] + '-');
            }
            document.write('<br>');
            let vec2 = vec.slice(1, 4);
            document.write('Array resultante de la llamada a slice<br>');
            for (let f = 0; f < vec2.length; f++) {
                document.write(vec2[f] + '-');
            }
        </script>
    </body>
</html>
```

3. Manipulación y operaciones con arrays

6) BÚSQUEDA DE ELEMENTOS EN UN ARRAYS

Se puede realizar de distintas formas: **indexOf(), lastIndexOf(), includes()**

a) indexOf()

Devuelve:

- El índice de la posición que ocupa el primer elemento que ha encontrado
- -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
```

3. Manipulación y operaciones con arrays

b) lastIndexOf()

Realiza las mismas tareas que indexOf pero trabajando desde el extremo derecho del array.

```
const animals = ['Dodo', 'Tiger', 'Penguin', 'Dodo'];

console.log(animals.lastIndexOf('Dodo'));
// Expected output: 3

console.log(animals.lastIndexOf('Tiger'));
// Expected output: 1
```

c) includes()

Se utiliza si no se necesita conocer la posición del elemento, sino sólo si el elemento existe en el array. Devuelve:

- True si encuentra al menos una coincidencia.
- False si no encuentra ninguna coincidencia.

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

3. Manipulación y operaciones con arrays

7) MÉTODO JOIN

El método join une todos los elementos del array en una única cadena.

A este método le pasamos como parámetro el o los caracteres de separación que debe agregar entre los elementos dentro del string que genera.

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <title>Ejemplo de JavaScript</title>
5           <meta charset="UTF-8">
6       </head>
7       <body>
8           <script>
9               let vec = [10, 20, 30, 40, 50];
10              let cadena = vec.join('-');
11              document.write(cadena);
12          </script>
13      </body>
14  </html>
```

10-20-30-40-50

El string cadena almacena todos los elementos del vector separados por el carácter que le hemos indicado (-).

Si llamamos al método join sin parámetros, se agrega por defecto el carácter de la coma.

3. Manipulación y operaciones con arrays

8) ORDENACIÓN DE ARRAYS

Unicode
https://symbl.cc/es/unicode/table/

La ordenación de arrays se realiza mediante el método **sort**.

Sort permite realizar cualquier ordenación que necesitemos.

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

Si en vez de números, los elementos del array son cadenas de caracteres, hay que tener en cuenta que ocurre algo inesperado:

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

La ordenación se realiza por orden alfabético, pero teniendo en cuenta el lugar que ocupa cada carácter en la tabla Unicode. Por esto, Casado, se ordena antes que casa. También ocurre lo mismo con la ñ que aparece en Unicode después del resto de letras.

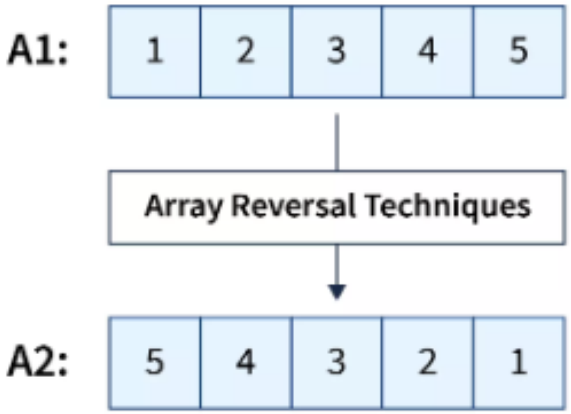
Para evitar estos contratiempos y modificar el comportamiento predeterminado de la ordenación, podemos establecer criterios propios, esto lo veremos con las funciones.

3. Manipulación y operaciones con arrays

Otro método útil es **reverse**, que invierte el orden de un 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']
```

Arrays
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array



4. Conjuntos

CONJUNTOS (Sets) – estructuras de datos similares a los arrays pero con la particularidad de que no permiten valores duplicados.

Para que los conjuntos no tengan elementos duplicados, no hay que hacer nada, ya que se hace automáticamente.

1. CREACIÓN DE CONJUNTOS

Para crear conjuntos se utiliza:

```
conjunto = new Set();
```

Para crear un conjunto, debemos hacerlo a partir de un elemento que podamos recorrer como: array, map, string, set.

Si queremos crear un conjunto e indicar los elementos que contiene se haría:

```
var conjunto1 = new Set([34,1,"Girasol",25.9]);
var conjunto2 = new Set("cadena");
```

```
▶ Set(4) {34, 1, 'Girasol', 25.9}
▶ Set(5) {'c', 'a', 'd', 'e', 'n'}
>
```

En estos ejemplos se ha creado un conjunto de un array y otro de un string. En el segundo caso, vemos como se eliminan automáticamente los elementos duplicados, en este caso la “a”.

Ejemplo 6: CONJUNTOS

Construye un conjunto a partir de un array que contenga los días de la semana. Luego muestra el conjunto creado por la consola del navegador.

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <title>Ejemplo de JavaScript</title>
5           <meta charset="UTF-8">
6       </head>
7       <body>
8           <script>
9               let dias=["Lunes", "Martes", "Miércoles", "Jueves", "Sábado", "Domingo"];
10              let conjuntoSemana = new Set(dias);
11              console.info(conjuntoSemana);
12          </script>
13      </body>
14  </html>
```

```
▶ Set(6) {'Lunes', 'Martes', 'Miércoles', 'Jueves', 'Sábado', …}
```

2. RECORRIDO DE CONJUNTOS

El recorrido de un conjunto se puede realizar con un bucle **for..of**:

```
var conjunto = new Set(["primero","segundo","tercero","primero"]);
for (let elemento of conjunto) {
        console.log(elemento);
}
```

```
primero
segundo
tercero
>
```

3. ADICIÓN DE ELEMENTOS A UN CONJUNTO

Para añadir elementos a un conjunto se utiliza el método **add**.

Para utilizar el método add, debemos indicar el elemento que deseamos añadir,

Add permite que las inserciones se encadenen.

```
var conjunto = new Set();
conjunto.add(7);
conjunto.add("Samuel").add(69).add("moteros");
// conjunto {7, 'Samuel', 69, 'moteros'}
```

4. ELIMINACIÓN DE ELEMENTOS DE CONJUNTOS

La eliminación de un elemento concreto del conjunto puede realizarse con el método **delete**, que devuelve true o false para indicar el resultado de la operación.

```
conjunto {7, 'Samuel', 69, 'moteros'}
 conjunto.delete(69);
 // conjunto {7, 'Samuel', 'moteros'}
```

Para eliminar todos los elementos de un conjunto, no es necesario llamar a delete tantas veces como elementos haya. Puede usarse el método **clear**.

```
conjunto.clear();
// conjunto {}
```

5. TAMAÑO DE UN CONJUNTO

TAMAÑO DE UN CONJUNTO – conocer el número de elementos que lo forman. Se utilizar la propiedad **size**.

```
var conjunto = new Set().add(1).add(1).add(2).add(9);
console.info(conjunto.size);
// 3
```

6. BÚSQUEDA DE UN ELEMENTO EN UN CONJUNTO

Con el método **has** se comprueba que un valor o el resultado de una expresión, están en el conjunto.

Devuelve un valor booleano para indicar si se encontró el elemento o no.

```
var conjunto = new Set().add(1).add(1).add(2).add(9);
if (conjunto.has(9))
        console.log("Encontrado");
// Encontrado
```

7. CONVERSIONES

Se puede convertir un conjunto en un array. Para ello utilizamos el operador de arrastre, de propagación o **spread**.

Se utiliza escribiendo unos puntos suspensivos … seguidos del nombre del conjunto que se desea convertir. Se devuelve el array resultado de la conversión.

```
var conjunto = new Set().add(1).add(1).add(2).add(9);
const vector = [...conjunto];
// vector [1, 2, 9]
```

Ejemplo 7: CONJUNTOS

Utiliza los conceptos de conjuntos para eliminar los elementos duplicados de un array y muestra los resultados por consola.

```
Contenido de array con duplicados:
▶ (6) [1, 7, 4, 7, 7, 2]
Conjunto creado a partir del array con duplicados:
▶ Set(4) {1, 7, 4, 2}
Array sin duplicados creado a partir del conjunto
▶ (4) [1, 7, 4, 2]
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ejemplo de JavaScript</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <script>
            let arrayConDuplicados = [1, 7, 4, 7, 7, 2];
            console.log('Contenido de array con duplicados:');
            console.log(arrayConDuplicados);
            let conjuntoArray = new Set(arrayConDuplicados);
            console.log('Conjunto creado a partir del array con duplicados:');
            console.log(conjuntoArray);
            let arraySinDuplicados = [...conjuntoArray];
            console.log('Array sin duplicados creado a partir del conjunto');
            console.log(arraySinDuplicados);
        </script>
    </body>
</html>
```

8. UNIÓN

A veces resulta útil crear un conjunto a partir de dos o más arrays que contienen elementos de interés.

Con el operador **spread** y la conversión automática de arrays en conjuntos eliminando los elementos duplicados se puede conseguir un gran ahorro de líneas de código.

```
let array1 = [10,20,30,40,50];
let array2 = [30,50,60,70,80];
let array3 = [60,70,80,90,100];
var conjunto = new Set([...array1,...array2,...array3]);
// conjunto {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
```

Conjuntos
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Set

5. Mapas

Los **mapas** permiten tener una colección de datos organizados en forma de pares "*clave-valor*", en las cuales, las **claves no se pueden repetir**.

Para crear un mapa utilizamos:

```
mapa = new Map();
```

Al igual que ocurre con los conjuntos, los valores iniciales de un mapa los puede aportar un array.,

```
const telefonos = new Map([
        [615885225,"Elena"],
        [663998541,"Quirós"],
        [656232511,"Marta"],
        [696585537,"David"]
]);
```

```
▶ 0: {615885225 => "Elena"}
▶ 1: {663998541 => "Quirós"}
▶ 2: {656232511 => "Marta"}
▶ 3: {696585537 => "David"}
size: 4
```

5. Mapas

1) RECORRIDO DE UN MAPA

Los mapas también se suelen recorrer con un bucle **for..of**. Pero debe utilizarse con cuidado.

```
const telefonos = new Map([
        [615885225,"Elena"],
        [663998541,"Quirós"],
        [656232511,"Marta"],
        [696585537,"David"]
]);
for (persona of telefonos)
        console.log(persona);
```

```
▶ (2) [615885225, 'Elena']
▶ (2) [663998541, 'Quirós']
▶ (2) [656232511, 'Marta']
▶ (2) [696585537, 'David']
>
```

Con este bucle se obtiene el par completo "*clave-valor*" de cada elemento del mapa. Esta salida no es demasiado útil para tratar los datos por separado.

Para obtener en distintas variables las claves y los valores de cada elemento se puede utilizar el siguiente bucle:

```
for (let [telefono,persona] of telefonos)
        console.log(`El teléfono de ${persona} es ${telefono}.`);
```

```
El teléfono de Elena es 615885225.
El teléfono de Quirós es 663998541.
El teléfono de Marta es 656232511.
El teléfono de David es 696585537.
>
```

5. Mapas

Si sólo estamos interesados en trabajar con las claves o con los valores se pueden usar los métodos **keys** y **values**.

- **keys()** – nos muestra las claves de un mapa.
- **values()** – nos muestra los valores de un mapa.

```
for (let telefono of telefonos.keys())
    console.log(telefono);
```

```
615885225
663998541
656232511
696585537
>
```

```
for (let persona of telefonos.values())
    console.log(persona);
```

```
Elena
Quirós
Marta
David
>
```

2) ADICCIÓN DE ELEMENTOS A UN MAPA

Para añadir elementos a un mapa, se utiliza el método **set**.

Al método **set**, le debemos indicar tanto la clave, como el valor del elemento.

Los mapas permiten encadenar las inserciones, pero hay que tener en cuenta que, si se añaden varios valores con la misma clave, sólo permanecerá en el mapa aquel valor que se haya insertado más tarde.

```
const telefonos = new Map();
telefonos.set(615885225,"Elena");
telefonos.set(615885225,"Elena").set(777777777,"Quirós");
telefonos.set(656232511,"Marta").set(777777777,"Sara");
telefonos.set(696585537,"David");
```

```
▶ 0: {615885225 => "Elena"}
▶ 1: {777777777 => "Sara"}
▶ 2: {656232511 => "Marta"}
▶ 3: {696585537 => "David"}
size: 4
```

En este ejemplo, vemos que se han añadido 6 elementos al mapa, pero en la salida, sólo aparecen correctamente insertados 4, ya que hay varios que repiten la clave. Esto se debe a la propia naturaleza de los mapas:

a) la clave correspondiente a Elena estaba duplicada por lo tanto sólo se muestra uno.

b) La clave de Quirós y Sara es la misma, pero se mantiene Sara al ser la última en introducirse.

3) ELIMINACIÓN DE LOS ELEMENTOS DE UN MAPA

Para eliminar los elementos de un mapa, se utiliza el método **delete**. Se debe indicar la clave del elemento que queremos eliminar.

```
▶ 0: {615885225 => "Elena"}
▶ 1: {777777777 => "Sara"}
▶ 2: {656232511 => "Marta"}
▶ 3: {696585537 => "David"}
size: 4
```

```
telefonos.delete(777777777);
```

```
▶ 0: {615885225 => "Elena"}
▶ 1: {656232511 => "Marta"}
▶ 2: {696585537 => "David"}
size: 3
```

4) BÚSQUEDA DE ELEMENTOS EN UN MAPA

Para buscar elementos en un mapa se utiliza el método has con la clave del elemento que se desea encontrar. Si lo encuentra devolverá true y si no, devolverá false.

```
▶ 0: {615885225 => "Elena"}
▶ 1: {656232511 => "Marta"}
▶ 2: {696585537 => "David"}
size: 3
```

```
if (telefonos.has(666555222))
        console.log("¡Encontrado!");
else
        console.log("No está.");

// No está.
```

5) LECTURA DE VALORES DE UN MAPA

Se utiliza el método **get**. Se introduce la clave como parámetro y da como resultado el valor asociado.

```
console.log(telefonos.get(656232511));
```

Ejemplo 8: MAPAS I

Crea un mapa vacío y añade los DNI de 5 personas ficticias. Se debe usar el DNI como calve y el nombre como valor.

Muestra por pantalla la lista de todos los DNIs junto con el nombre de las personas.

Modifica el nombre de la tercera persona y vuelve a mostrar todos los datos en pantalla para comprobar que la operación se ha realizado correctamente.

```
LISTADO INICIAL
NOMBRE: Juan - DNI: 12345678A
NOMBRE: María - DNI: 23456789B
NOMBRE: Pedro - DNI: 34567891C
NOMBRE: Laura - DNI: 45678912D
NOMBRE: Luis - DNI: 56789123E

LISTADO MODIFICADO
NOMBRE: Juan - DNI: 12345678A
NOMBRE: María - DNI: 23456789B
NOMBRE: Daniel - DNI: 34567891C
NOMBRE: Laura - DNI: 45678912D
NOMBRE: Luis - DNI: 56789123E
```

5. Mapas

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ejemplo de JavaScript</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <script>
            let personas = new Map();
            //Añadir los DNIs y nombres de las personas al mapa
            personas.set("12345678A", "Juan");
            personas.set("23456789B", "María");
            personas.set("34567891C", "Pedro");
            personas.set("45678912D", "Laura");
            personas.set("56789123E", "Luis");
            //Mostrar por pantalla el DNI y nombres de las personas
            document.write("LISTADO INICIAL<br><br>");
            for (let[dni,nombre] of personas){
                document.write('NOMBRE: '+ nombre + ' - DNI: '+ dni +'<br><br>');
            }
            //Modificar el nombre de la tercera persona
            let dniArray = Array.from(personas.keys());
            personas.set(dniArray[2], "Daniel");
            //Mostrar los DNI y nombres actualizados
            document.write("<BR>LISTADO MODIFICADO<br><br>");
            for (let[dni,nombre] of personas){
                document.write('NOMBRE: '+ nombre + ' - DNI: '+ dni +'<br><br>');
            }
        </script>
    </body>
</html>
```

LISTADO INICIAL

NOMBRE: Juan - DNI: 12345678A

NOMBRE: María - DNI: 23456789B

NOMBRE: Pedro - DNI: 34567891C

NOMBRE: Laura - DNI: 45678912D

NOMBRE: Luis - DNI: 56789123E

LISTADO MODIFICADO

NOMBRE: Juan - DNI: 12345678A

NOMBRE: María - DNI: 23456789B

NOMBRE: Daniel - DNI: 34567891C

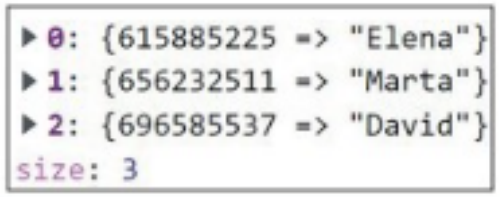
NOMBRE: Laura - DNI: 45678912D

NOMBRE: Luis - DNI: 56789123E

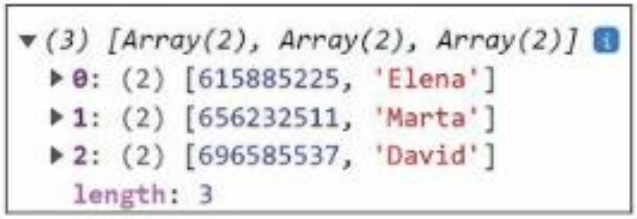
6) CONVERSIONES

Es posible obtener un array a partir del contenido de un mapa. Para hacerlo, se usa el mismo operador que con los conjuntos **spread**.

Con spread obtenemos un array de arrays.




```
console.log([...telefonos]);
```



Mapas
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

Gracias