



DESARROLLO WEB EN ENTORNO CLIENTE

# UD4 – ARRAYS

---

CICLO FORMATIVO DE GRADO SUPERIOR EN DESARROLLO DE  
APLICACIONES WEB

I.E.S. HERMENEGILDO LANZ – 2022/2023

PROFESORA: VANESA ESPÍN

[vespin@ieshlanz.es](mailto:vespin@ieshlanz.es)



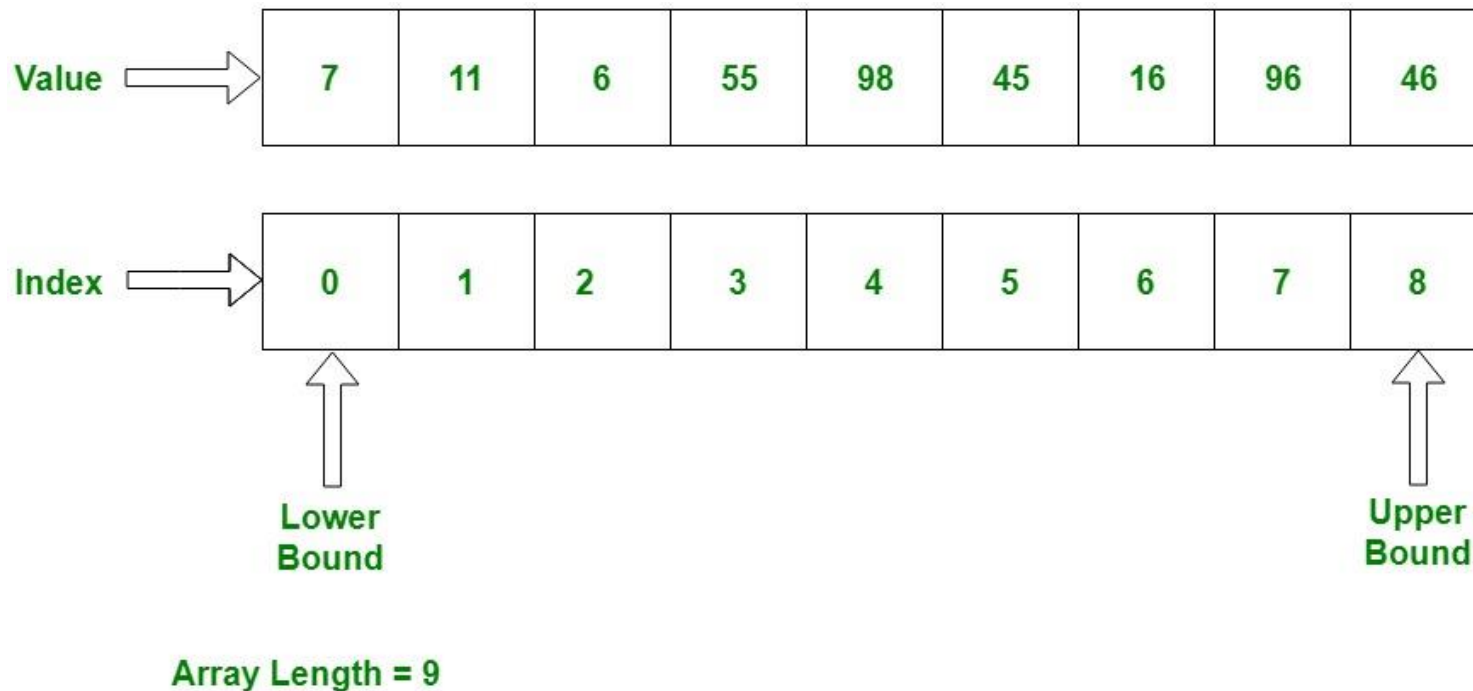
# Índice

---

1. Estructuras de tipo Array
  - a. Creación
  - b. Operaciones
  - c. Propiedades y métodos
  - d. Recorrido e iteración
  - e. Arrays paralelos
  - f. Arrays Multidimensionales
2. Estructuras de tipo SET
3. Estructuras de tipo MAP
4. Funciones
5. Objetos

# ARRAYS

- En programación, **un array se define como una colección ordenada de datos.**
- RECUERDA: JavaScript emplea arrays internamente para gestionar objetos HTML en el documento, propiedades del navegador, etc. Por ejemplo, si tu documento contiene 10 enlaces: `document.links[0]` , representará el primer enlace en el documento.



OJO: los arrays en Javascript son heterogéneos:  
`A=[3,4,"pepe",true,Math.random()]`

# Creación de Arrays

- Seguiremos por: [https://www.w3schools.com/js/js\\_arrays.asp](https://www.w3schools.com/js/js_arrays.asp)
- Arrays como literales

```
const cars = ["Saab", "Volvo", "BMW"];
```

```
const cars = [];  
cars[0] = "Saab";  
cars[1] = "Volvo";  
cars[2] = "BMW";
```

```
const cars = [  
  "Saab",  
  "Volvo",  
  "BMW"  
];
```

- Arrays como objetos

```
const cars = new Array("Saab", "Volvo", "BMW");
```

The two examples above do exactly the same.

There is no need to use `new Array()`.

For simplicity, readability and execution speed, use the array literal method.

# Operaciones con Arrays

- Acceso a elementos

```
const cars = ["Saab", "Volvo", "BMW"];  
let car = cars[0];
```

- Cambiando un elemento

```
cars[0] = "Opel";
```

- Accediendo al array completo

```
const cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;
```

- Los arrays **son objetos que también pueden contener elementos objeto**

```
myArray[0] = Date.now;  
myArray[1] = myFunction;  
myArray[2] = myCars;
```

# Operaciones con Array

- **Length**: longitud del array. Número de elementos

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let length = fruits.length;
```

- Acceso a primer y ultimo elemento:

```
let fruit = fruits[0];  
let fruit = fruits[fruits.length - 1];
```

- Añadir elementos con **push**

```
fruits.push("Lemon"); // Adds a new element (Lemon) to fruits
```

- Es igual que añadirlo al final

```
fruits[fruits.length] = "Lemon"; // Adds "Lemon" to fruits
```

**WARNING !** `fruits[6] = "Lemon"; // Creates undefined "holes" in fruits`

Adding elements with high indexes can create undefined "holes" in an array:



# ARRAYS como objetos?

- Los **arrays asociativos (o hashes)** son arrays cuyos índices tienen “nombre”. Algunos lenguajes de programación los permiten pero Javascript NO. En Javascript siempre usamos índices numéricos.

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

Arrays are a special kind of objects, with numbered indexes.

- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

```
const points = new Array();  
const points = [];
```

A Common Error  
OJO NO ES LO MISMO!!!

```
const points = [40]; //array de un elemento
```

is not the same as:

```
const points = new Array(40); //array de 40 elementos
```

# Cómo reconocer un array

```
const fruits = ["Banana", "Orange", "Apple"];  
let type = typeof fruits;
```

→ **typeof** Devuelve object. ¿Cómo sé que es array?

## SOLUCIÓN 1:

```
Array.isArray(fruits);
```

## SOLUCIÓN 2:

```
const fruits = ["Banana", "Orange", "Apple"];  
  
fruits instanceof Array;
```



# Arrays. Métodos y Propiedades

- Todos los métodos y propiedades: [https://www.w3schools.com/jsref/jsref\\_obj\\_array.asp](https://www.w3schools.com/jsref/jsref_obj_array.asp)

Métodos	Descripción
<b>concat()</b>	Une dos o más arrays, y devuelve una copia de los arrays unidos.
<b>join()</b>	Une todos los elementos de un array en una cadena de texto.
<b>pop()</b>	Elimina el último elemento de un array y devuelve ese elemento.
<b>push()</b>	Añade nuevos elementos al final de un array, y devuelve la nueva longitud.
<b>reverse()</b>	Invierte el orden de los elementos en un array.
<b>shift()</b>	Elimina el primer elemento de un array, y devuelve ese elemento.
<b>slice()</b>	Selecciona una parte de un array y devuelve el nuevo array.
<b>sort()</b>	Ordena los elementos de un array.
<b>splice()</b>	Añade/elimina elementos a un array.
<b>toString()</b>	Convierte un array a una cadena y devuelve el resultado.
<b>unshift()</b>	Añade nuevos elementos al comienzo de un array, y devuelve la nueva longitud.
<b>valueOf()</b>	Devuelve el valor primitivo de un array.

# Arrays. Métodos y Propiedades

Propiedad	Descripción
<b>constructor</b>	Devuelve la función que creó el prototipo del objeto array.
<b>length</b>	Ajusta o devuelve el número de elementos en un array.
<b>prototype</b>	Te permite añadir propiedades y métodos a un objeto.

**ESTUDIAREMOS MÉTODOS Y PROPIEDADES POR:**

[https://www.w3schools.com/js/js\\_array\\_methods.asp](https://www.w3schools.com/js/js_array_methods.asp)

# Ordenar Arrays


[https://www.w3schools.com/js/js\\_array\\_sort.asp](https://www.w3schools.com/js/js_array_sort.asp)

- sort ordena alfabéticamente con orden creciente
- reverse ordena alfabéticamente con orden decreciente

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();  
fruits.reverse();
```

- Ojo sort y reverse no ordenan bien los números podemos usar una función de comparación

```
const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a - b});  
points.sort(function(a, b){return b - a});
```

- Ordenar por atributo de objeto:  *alfabéticamente*  

```
cars.sort(function(a, b){return a.year - b.year});
```
- No hay métodos para hallar el mayor o menor. Opciones:
  - 1) Los ordenamos y cogemos el primero o el último.
  - 2) Usamos Math.max.apply o Math.min.apply  

```
return Math.max.apply(null, arr);
```

```
cars.sort(function(a, b){  
  let x = a.type.toLowerCase();  
  let y = b.type.toLowerCase();  
  if (x < y) {return -1;}  
  if (x > y) {return 1;}  
  return 0;  
});
```

# Recorrer Arrays

for, for..in y for..of

```
for (let i=0;i<sistemaSolar.length;i++){  
    console.log(`planeta ${i} es ${sistemaSolar[i]}`);  
}  
  
for (let i in sistemaSolar){  
    console.log(`planeta ${i} es ${sistemaSolar[i]}`);  
}  
  
for (let planeta of sistemaSolar){  
    console.log(planeta);  
}
```

```
var sistemaSolar = new Array();  
sistemaSolar[0] = "Mercurio";  
sistemaSolar[1] = "Venus";  
sistemaSolar[2] = "Tierra";  
sistemaSolar[3] = "Marte";  
sistemaSolar[4] = "Jupiter";  
sistemaSolar[5] = "Saturno";  
sistemaSolar[6] = "Urano";  
sistemaSolar[7] = "Neptuno";
```

# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **forEach:** entra dentro del grupo de Iterables y ejecuta una función por cada elemento del array. En cada iteración se tendrá acceso a 3 variables: valor (del elemento), índice (del elemento) y array (que estamos recorriendo). Muy útil cuando solo necesitamos ejecutar una función a través de cada elemento del arreglo, sin necesidad de obtener un retorno.

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt += value + "<br>";
}
```

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

```
const cars = ['Ferrari 250 GT Berlinetta.',
'Tesla S', 'Génesis G90', 'Porsche Boxster'];
```

```
//Con una función de devolución ES5
cars.forEach(function (element) {
  console.log(element);
});
```

```
//Función de flecha ES6
cars.forEach(element => console.log(element));
```

# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **Array.map():** entra dentro del grupo de Iterables y ejecuta una función por cada elemento del array. Te permite iterar sobre un arreglo y modificar sus elementos EN UN NUEVO ARRAY utilizando una función callback.

```
const numbers1 = [45, 4, 9, 16, 25];  
const numbers2 = numbers1.map(myFunction);  
  
function myFunction(value, index, array) {  
  return value * 2;  
}
```

Podemos omitir los parámetros que no usemos:

```
const numbers1 = [45, 4, 9, 16, 25];  
const numbers2 = numbers1.map(myFunction);  
  
function myFunction(value) {  
  return value * 2;  
}
```

# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **Array.filter():** entra dentro del grupo de Iterables y ejecuta una función por cada elemento del array. Te permite iterar sobre un arreglo y crear UN NUEVO ARRAY utilizando una función callback para quedarnos con los elementos que pasen un test

Nos quedamos con los mayores de 18:

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Podemos omitir los parámetros que no usemos:

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

function myFunction(value) {
  return value > 18;
}
```



# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **Array.indexOf():** entra dentro del grupo de Iterables y busca en un array un elemento devolviendo la posición en que aparece por primera vez. -1 si no lo encuentra

SINTAXIS: `array.indexOf(item, start)`

<i>Item</i>	Required. The item to search for.
<i>start</i>	Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the end.

Buscamos "Apple". Devuelve posición:

```
const fruits = ["Apple", "Orange", "Apple", "Mango"];  
let position = fruits.indexOf("Apple"); //Devuelve 0  
let position = fruits.indexOf("Apple",1); //Devuelve 2
```

# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **Array.lastIndexOf():** entra dentro del grupo de Iterables y busca en un array un elemento devolviendo la posición en que aparece por última vez. -1 si no lo encuentra

SINTAXIS: `array.lastIndexOf(item, start)`

<i>Item</i>	Required. The item to search for.
<i>start</i>	Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the beginning

Buscamos "Apple". Devuelve posición:

```
const fruits = ["Apple", "Orange", "Apple", "Mango"];  
let position = fruits.lastIndexOf("Apple"); //Devuelve 2  
let position = fruits.lastIndexOf("Apple",-3); //Devuelve 0
```

# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **Array.find():** entra dentro del grupo de Iterables y ejecuta una función por cada elemento del array. Devuelve el valor del primera elemento que supera un test. -1 si no lo encuentra.

Devuelve el primer elemento mayor de 18:

```
const numbers = [4, 9, 16, 25, 29];  
let first = numbers.find (myFunction); //Devuelve 25  
  
function myFunction(value, index, array) {  
    return value > 18;  
}
```

No es soportado por Internet Explorer  
pero sí por el resto de los navegadores  
modernos

# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **Array.findIndex():** entra dentro del grupo de Iterables y ejecuta una función por cada elemento del array. Devuelve la posición del primera elemento que supera un test. -1 si no lo encuentra.

Devuelve la posición del primer elemento mayor de 18:

```
const numbers = [4, 9, 16, 25, 29];  
let first = numbers.findIndex(myFunction); //Devuelve 3  
  
function myFunction(value, index, array) {  
    return value > 18;  
}
```

No es soportado por Internet Explorer  
pero sí por el resto de los navegadores  
modernos

# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **Array.from():** devuelve un objeto **Array** a partir de un objeto iterable o con propiedad length.

```
Array.from("ABCDEFGH");
```

- **Array.keys():** devuelve un objeto **Array Iterator** con las claves (posiciones) de un array

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
const keys = fruits.keys();  
  
for (let x of keys) {  
  text += x + "<br>";  
}
```

# Iteración en Arrays

[https://www.w3schools.com/js/js\\_array\\_iteration.asp](https://www.w3schools.com/js/js_array_iteration.asp)

- **Array.entries():** devuelve un objeto **Array Iterador** con los pares clave/valor de un array

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const f = fruits.entries();

for (let x of f) {
  document.getElementById("demo").innerHTML += x;
}
```

- **Array.includes():** chequea si un elemento está incluido en un array. Devuelve true o false.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.includes("Mango"); // is true
```

# Desestructuración de Arrays

Usado en React

## Before:

```
const vehicles = ['mustang', 'f-150', 'expedition'];

// old way
const car = vehicles[0];
const truck = vehicles[1];
const suv = vehicles[2];
```

Here is the new way of assigning array items to a variable:

## With destructuring:

```
const vehicles = ['mustang', 'f-150', 'expedition'];

const [car, truck, suv] = vehicles;
```



# Arrays Paralelos

- Cuando tenemos dos o más arrays, que utilizan el mismo índice para referirse a términos homólogos, se denominan **arrays paralelos**.
- Para que los arrays paralelos sean homogéneos, éstos tendrán que tener la misma longitud (para mantener estructura lógica). Nos permitirán búsquedas en varios arrays a la vez en arrays que estén sincronizados.
- Entre las **ventajas del uso de arrays paralelos** tenemos:
  - Son fáciles de entender y utilizar.
  - Pueden ahorrar una gran cantidad de espacio y evitar complicaciones de sincronización.
  - El recorrido secuencial de cada posición del array, es extremadamente rápido en las máquinas actuales.

ejemplo

# Arrays Paralelos

```
var profesores = ["Cristina","Catalina","Vieites","Benjamin"];
var asignaturas=["Seguridad","Bases de Datos","Sistemas Informáticos","Redes"];
var alumnos=[24,17,28,26];

function imprimeDatos(indice)
{
    document.write("<br/>" + profesores[indice] + " del módulo de " + asignaturas[indice] + ", tiene " + alumnos[indice] + " alumnos en clase.");
}

for (i=0;i<profesores.length;i++)
{
    imprimeDatos(i);
}
```

## RESULTADO:

```
Cristina del módulo de Seguridad, tiene 24 alumnos en clase.
Catalina del módulo de Bases de Datos, tiene 17 alumnos en clase.
Vieites del módulo de Sistemas Informáticos, tiene 28 alumnos en clase.
Benjamin del módulo de Redes, tiene 26 alumnos en clase.
```

# Arrays Multidimensionales

- Una alternativa a los arrays paralelos es la simulación de un array multidimensional. Podemos crear arrays que en sus posiciones contengan otros arrays u otros objetos. Podemos crear de esta forma *arrays bidimensionales*, *tridimensionales*, etc.
- Por ejemplo podemos realizar el ejemplo anterior creando un **array bidimensional** así:

## Mediante objeto array

```
var datos = new Array();
datos[0] = new Array("Cristina","Seguridad",24);
datos[1] = new Array("Catalina","Bases de Datos",17);
datos[2] = new Array("Vieites","Sistemas Informáticos",28);
datos[3] = new Array("Benjamin","Redes",26);
```

## Mediante literal

```
var datos = [
    ["Cristina","Seguridad",24],
    ["Catalina","Bases de Datos",17],
    ["Vieites","Sistemas Informáticos",28],
    ["Benjamin","Redes",26]
];
```

# Arrays Multidimensionales

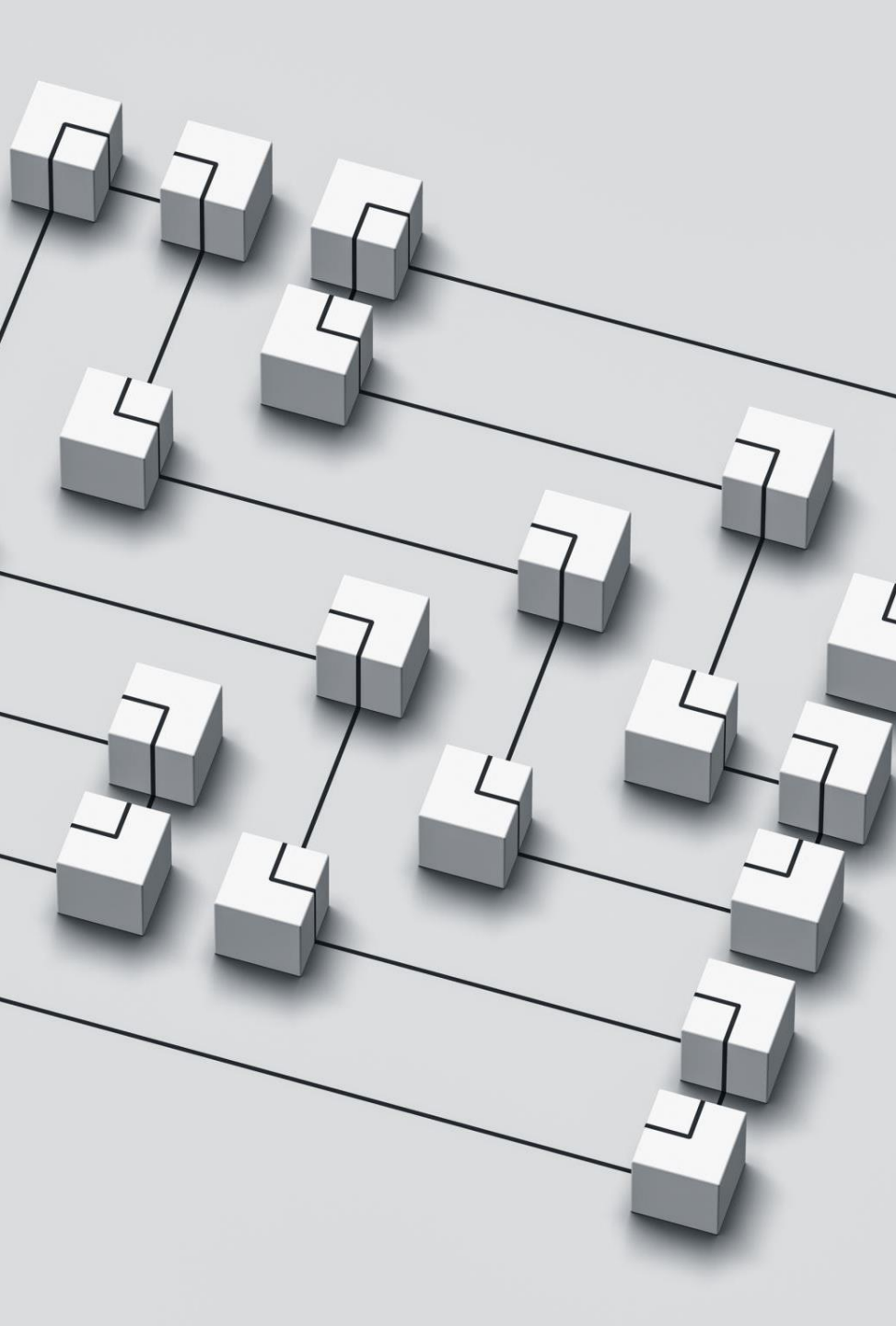
- Para acceder a un dato de un array de arrays, se hará escribiendo el nombre del array, y entre corchetes cada una de las referencias: `array[i][j]` (nos permitiría acceder a una posición determinada en un array bidimensional).

```
document.write("<br/>Quien imparte Bases de Datos? "+datos[1][0]); // Catalina
document.write("<br/>Asignatura de Vieites: "+datos[2][1]);           // Sistemas Informaticos
document.write("<br/>Alumnos de Benjamin: "+datos[3][2]);           // 26
```

Recorrido de dimensiones. Por ejemplo:

```
for (i=0;i<datos.length;i++)
{
    document.write("<br/>" + datos[i][0] + " del módulo de "
    + datos[i][1] + ", tiene " + datos[i][2] + " alumnos en clase.");
}
```

```
Cristina del módulo de Seguridad, tiene 24 alumnos en clase.
Catalina del módulo de Bases de Datos, tiene 17 alumnos en clase.
Vieites del módulo de Sistemas Informáticos, tiene 28 alumnos en clase.
Benjamin del módulo de Redes, tiene 26 alumnos en clase.
```



## 2. SET (CONJUNTOS)

---

- Son una Estructura de Datos, que, de forma similar a los arrays **permiten almacenar datos**.
- Al igual que los arrays: son objetos.
- Principal virtud: **no permiten valores repetidos**.
- **Declaración e inicialización** de un conjunto vacío:

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

# Añadir valores a un SET

- Una vez declarado e inicializado el conjunto, podemos añadirle valores mediante el método ***add***:

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

```
conjunto.add(8);  
conjunto.add(6);  
conjunto.add(5);  
conjunto.add(5);  
conjunto.add(6);
```

otra forma

```
conjunto.add(8).add(6).add(5).add(5).add(6);
```

Vemos cómo se han eliminado los valores repetidos.  
HACER ESTO CON ARRAYS ES MUCHO MÁS ENGORROSO.

PROBLEMAS	SALIDA	CONSOLA DE DEPURACIÓN	TERMINAL
-----------	--------	-----------------------	----------

```
> Set(3) {size: 3, 8, 6, 5}
```

```
console.log(conjunto);
```

# Añadir valores a un SET

- También puedo iniciar el conjunto con un **array**:

```
const conjunto=new Set( [5,6,4,5,6,5,4,6,4,4,5] ); //Set {5, 6, 4}
```

- O con una **cadena** (ojo, me crea un conjunto de caracteres)

```
const conjunto=new Set("Conjunto"); //Set {C, o, n, j, u , t}
```

- Si quiero añadir **Strings (en lugar de caracteres)** lo hago con **add**

```
const conjunto=new Set();  
conjunto.add("Conjunto");  
console.log(conjunto); //Set {'Conjunto'}
```



# Propiedades y Métodos de SET

- Tamaño (número de elementos): `conjunto.size;`
- Eliminar valores (método ***delete***): `conjunto.delete(6); //Set {5, 4}`
- Vaciar el conjunto (método ***clear***): `conjunto.clear(); //Set { }`
- Buscar valores (método ***has***): `conjunto.has(5); //true` `conjunto.has(6); //false`
- Convertir en `Array`: (Operador de **propagación o spread**):

***[... elem a propagar]***

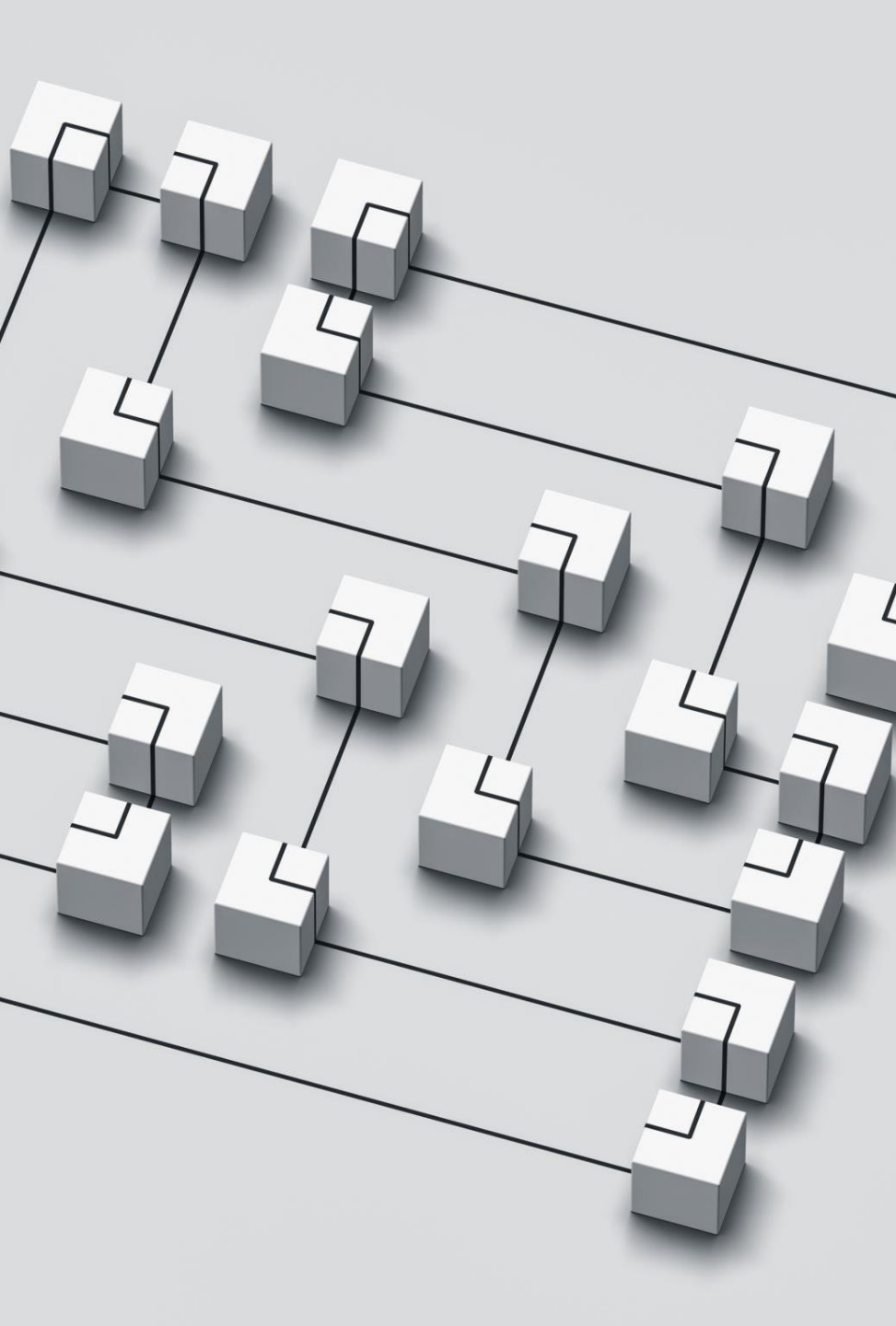
```
const conjunto=new Set([1,3,5,7]);  
const miarray = [...conjunto];  
console.log(miarray); // [1, 3, 5, 7]
```

El conjunto se ha  
convertido en array

# Recorrido de Conjuntos

EN LOS CONJUNTOS NO  
EXISTE EL CONCEPTO DE  
*POSICIÓN DE ELEMENTO*

```
const conjunto=new Set([1,3,5,7]);  
  
// "for of"  
for (let item of conjunto) console.log(item);  
  
// "for of con values"  
for (let item of conjunto.values())  
  console.log(item);  
  
// "forEach" con función anónima callback  
conjunto.forEach(function(value) {  
    console.log(value)  
});
```



### 3. MAP (MAPAS)

---

- Son una Estructura de Datos que permiten almacenar datos de tipo **clave-valor**.
- Característica: **no permiten claves repetidas**.
- Al igual que los arrays y los conjuntos: son objetos.
- **Declaración e inicialización** de un mapa vacío:

```
const provincias=new Map();
```

# Asignar valores a MAPAS

- Puedo hacerlo en la **inicialización** con un **array**:

Vemos que es como un array de estructuras clave-valor

```
const fruits = new Map( [
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
```

```
Map(3) {size: 3, apples => 500, bananas => 300, oranges => 200}
```

- Puedo usar el método **set (clave, valor)** para añadir un elemento:

También podría encadenar los set

```
// Create a Map
const fruits = new Map();
// Set Map Values
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);
```

# Propiedades y Métodos de MAP

- Tamaño del mapa `fruits.size;`
- Obtener valores con método ***get (clave)***. Devuelve el valor de esa clave:

```
console.log(fruits.get("apples")); // devuelve 500
```

- Para buscar una clave uso el método ***has (clave)***. Devuelve *true* o *false*:

```
console.log(fruits.has("apples")); // devuelve true
```

```
console.log(fruits.has("500")); // devuelve false
```

- Para borrar un elemento uso el método ***delete (clave)***.

```
fruits.delete("apples");
```

# Objetos Iterables de MAPAS

- Nos servirán para recorrer mediante bucles de tipo ***for .. of***
- Método ***keys()*** para obtener claves y ***values()*** para valores

```
let texto = "";
let frutas = fruits.keys();
for (const fruta of frutas) {
  texto += " "+fruta;
}
console.log (texto); //apples bananas oranges
```

```
let texto = "";
for (const cant of fruits.values()) {
  texto += " "+cant;
}
console.log (texto); //500 300 200
```

- Método ***entries()*** para obtener los pares clave-valor

```
let texto = "";
for (const par of fruits.entries()) {
  texto += par + "<br>";
}
```

```
console.log:
  apples,500
  bananas,300
  oranges,200
```

# Recorrido de MAPAS

- Podemos usar **for .. of** sin iterables para recorrer mapas

```
let texto = "";  
for (let fruta of fruits) {  
  texto += fruta + "\n";  
}  
console.log(texto);
```

apples,500  
bananas,300  
oranges,200

- También podemos **desestructurar** el array interior para separar en dos variables clave y valor:

```
for (let [fruta,cantidad] of fruits) {  
  console.log(`Clave: ${fruta}, Valor: ${cantidad}`);  
}
```

Clave: apples, Valor: 500
Clave: bananas, Valor: 300
Clave: oranges, Valor: 200