DW - Entorno Cliente

▼ 📁 Tema 3

▼ 📁 Arrays

▼ 📄 1. Creación y acceso

Crea array vacío:

- 3 formas:

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

Crea array con elementos:

- 3 formas:

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

De la misma forma se puede hacer con corchetes y sus valores:

```
let array1 = [2]
let array2 = [1,3]
let arrat3 = ["Luis"]
```

▼ 📄 2. Valores del array

Cambiar valores

- nombre del array y la posición del valor:

```
edades[0] = 111;
edades[2] = 998
```

Mostrarlos por pantalla:

- con `document.write();` para el navegador

- con `console.log();` para la consola del navegador → ==Ctrl + Shift + i==
- Con ambos seleccionar la posición que se quiere dentro del array
- Los espacios entre comas los ==rellena con elementos vacíos==
  - Ejemplo:

`let procesadores = ["Intel",,"AMD"]` → 2º vacío

`let precesadores = [,"Intel",,"AMD",]` → 1º y 3º vacío

▼ 📄 3. Arrays Bidimensionales

Creación

- Creación de un array bidimensional de 2 filas y 3 columnas

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

- Introducir datos:

```
tablaNotas[0][0] = 1;
tablaNotas[0][1] = 2;
tablaNotas[0][2] = 3;
...
```

Para recorrer los arrays bidimensionales se utilizan 2 bucles for anidados

▼ 📄 4. Recorrido de un Array

> Recorrer cada elemento del array
> Método para saber el total del elementos o longitud del array: `.length`

▼ Bucle FOR:

```
for (let i=0; i<precios.length; i++) {
    console.log("El precio " + i + "es: " + precios[i])
}
```

> este bucle imprime los valores vacíos

▼ Bucle FOR IN:

```
for (let elemento in precios) {
```

```
    console.log("El precio " + elemento + "es: " + precios[elemento])
}
```

este bucle no imprime los valores vacíos
no necesita inicializar el contador, ni controlar el tamaño del array

▼ Bucle FOR OF:

```
for (let precio of precios) {
    console.log(precio)
}
```

> Simplifica los otros dos for anteriores
> No se utiliza una variable para recorrer el array, se realiza automático
> Desventaja: se desconocen los los índices de las posiciones
> Imprime los valores vacíos

▼ 📄 5. Añadir elementos a un Array

Diferentes formas:

1. Sabiendo cuantos elementos tiene un array, se le indica la posición para el nuevo valor:

   ```
   let elementos = ["a", 7, true]
   elementos[3] = 22.45
   ```

   //Resultado `["a", 7, true, 23.45]`

2. Se desconoce el número de elementos. Utilizamos **push,** añade al final

   ```
   elementos.push("xyz");
   ```

3. Añadir al principio con **unshift**

   ```
   elementos.unshift("primero")
   ```

   > unshift y push permiten añadir varios valores

▼ 📄 6. Eliminación de elementos de un Array

1. Modificando
   - shift = elimina el primer elemento

- pop = elimina el último elemento

Ambos devuelven el elemento eliminado, de no haber devuelve **undefined**

2. Modificando la longitud del array
   - `elementos.length = 2;`
3. Splice
   - 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"]
```

   - Elimina los dos primeros elementos después de la posición 3, es decir, los elementos en 4 y 5.

▼ 📄 7. Concatenación de Arrays

Concat permite extender un array añadiéndole al final el contenido de otro array.
No se modifican los antiguos, sino que crea uno nuevo con la unión

▼ 📄 8. Copia de Arrays

Se pueden copiar completos o copiar parte de ellos con el método **slice**
Si no se indican parámetros al método se copia todo el array en otro array nuevo

`elementos.slice(2,4)`
- El primer parámetro es el índice de la posición desde la que se desea copiar (esa posición incluida)
- El segundo parámetro es el índice hasta el que se quiere copiar (no se incluye esta posición)

▼ 📄 9. Búsqueda de elementos en un Array

1. **indexOf()**

- El índice de la posición que ocupa el primer elemento que ha encontrado
- 1 si no 10 ha encontrado

2. **lastIndexOf()**
    - Empieza desde el extremo derecho del array y devuelve el último índice que se encuentre del elemento
3. **includes()**
    - Se utiliza si no interesa conocer la posición del elemento
    - Solo saber si existe dentro del array
    - Devuelve true o false

▼ 📄 10. Método Join

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

Se utiliza normalmente para dar resultados al imprimir.

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

▼ 📄 11. Ordenación de Arrays

Método **sort**

- Números en orden
- Cadenas de texto y caracteres depende de su primera letra en el código /unicode no orden alfabético si encuentra mayúsculas y minúsculas por ejemplo.

Método **reverse**

- Hace lo contrario

▼ 📁 Conjuntos

▼ 📄 1. Creación de Conjuntos

- Los conjuntos (sets) no permiten valores duplicados, aunque funcionan parecido a los arrays.
- Crear un nuevo conjunto vacío: `conjunto = new Set();`
- Crear un nuevo conjunto con datos, parte de arrays, mapas o strings:

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

▼ 📄 2. Valores del Conjunto

Se hace con el bucle **FOR...OF**

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

- en este caso el valor repetido "primero" dentro del array de inicio dentro de la declaración del conjunto, se quita el valor repetido al funcionar dentro de un conjunto

▼ 📄 3. Añadir elementos a un Conjunto

- Método add indicando el valor que se va a añadir
  - Se pueden hacer varios add al mismo tiempo

`conjunto.add()`

Para recorrer los arrays bidimensionales se utilizan 2 bucles for anidados

▼ 📄 4. Borrar elemdnos de un Conjunto

- Dos métodos:

1. Método delete
`conjunto.delete(7)`

2. Método clear
Deja el conjunto vacío
`conjunto.clear()`

📄 5. Tamaño del conjunto

Método `.size`

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

📄 6. Búsqueda de un elementos en Conjuntos

- Método `.has`

→ Devuelve un booleano

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

📄 7. Conversiones

- De conjunto a array. Se usa `spread`

Se devuelve un array con el resultado de la conversión

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

📄 8. Unión

- `Spread` + conversión a arrays

```
var conjunto = new Set([...array1,...array2,...array3]);
```

📁 Mapas

📄 1. Creación de Mapas

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

- Colección de datos organizados por `clave-valor`

Las claves no se pueden repetir

```
const frutas = new Map([
    ["A", "Manzana"],
    ["B", "Plátano"],
```

```
    ["C", "Naranja"],
    ["D", "Uva"]
]);
```

▼ 📄 2. Recorrido de un Mapa

También usan el bucle **FOR...OF**

```
for (persona of telefonos) {
    console.log(persona);
}
```
Con este bucle se obtiene el par completo "clave-valor" de cada elemento del mapa.

Se pueden imprimir por separado:

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

- con los métodos `keys()` y `values()` se pueden obtener las clave y valor por separado

```
for (let telofono of telefonos.keys())
    console.log(telefono)
for (let persona of telefonos.values())
    console.log(persona)
```

▼ 📄 3. Añadir y eliminar elementos a un Mapa

- Método `set`. A este método se le debe incluir tanto la clave como el valor del elemento
- Se pueden añadir varios elementos a la vez pero si la clave se repite solo persiste la última, si las claves se repiten en una adición posterior, sustituye por el nuevo valor.

```
telefonos.set(615885225, "Elena")
```

- Método `delete`

Indicar la clave del método que se quiere eliminar

```
telefonos.delete(123456789)
```

▼ 📄 4. Búsqueda de elementos en un Mapa

- Método .has con la clave

Devuelve boolean

```
if (telefonos.has(123456789))
    console.log("Encontrado"):
else
    console.log("No está");
```

- Lectura con el método .get

Necesita la clave como parámetro y devuelve el valor

```
console.log(telefonos.get(123456789));
```

▼ 📄5. Conversión de Mapas

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])
```