

## Guía teórico practica de arrays – map - set

### Arrays

Los objetos te permiten almacenar colecciones de datos a través de nombres. Pero a menudo necesitamos una **colección ordenada**, donde tenemos un 1ro, un 2do, un 3er elemento y así sucesivamente. Por ejemplo, necesitamos almacenar una lista de algo: usuarios, bienes, elementos HTML, etc.

un Array comienza y termina con corchetes [ ]

Crea un Array con tres elementos : 'Apple', 'Orange', y 'Plum'.

```
let fruits = ["Apple", "Orange", "Plum"];
```

Las comas para separación de elementos y es ignorado las comas finales.

```
console.log(fruits[1]);
```

Para leer un elemento Array, ponemos un índice entre corchetes (los índices comienzan en cero)

```
fruits[0] = 'pear';
```

Para cambiar un elemento de Array, asignamos a un Array con un índice:

```
console.log(fruits);
```

### length - push

```
1 let fruits = ["Apple", "Orange", "Plum"];
2 console.log(fruits.length); // 3
3
4 fruits[fruits.length] = "grape";
5 console.log(fruits); // [ 'Apple', 'Orange', 'Plum', 'grape' ]
6
7 fruits.push("banana");
8 console.log(fruits); // [ 'Apple', 'Orange', 'Plum', 'grape', 'banana' ]
9
10 fruits.length = 1;
11 console.log(fruits); // [ 'Apple' ]
12
```

Cada Array tiene una propiedad .length que se puede usar tanto para leer como para cambiar (!) el número de elementos en un Array.

Si escribimos en el Array en el índice de la longitud, agregamos un elemento:

Otra forma de agregar un elemento es a través del método Array .push()

Podando el Array se eliminan elementos

### array.at() - vaciar array

```
1 let fruits = ["Apple", "Orange", "Plum", "banana"];
2 console.log(fruits.at(index: -2)); // Plum
3
4 fruits = [];
5 console.log(fruits); // []
```

El método Array .at() devuelve el elemento en un índice dado. Admite índices positivos y negativos

Para borrar (vaciar) una matriz, podemos establecerla .length en cero o poner el array vacío.

Dentro de un array, un elemento de propagación o difusión consta de tres puntos ( ...) seguidos de una expresión.

```
1  people = ["Arle", "Juliana", "valentina", "Ana"];
2  people2 = ["Diego", "Juan"];
3
4  copy = [...people, ...people2, "Pepe"];
5  console.log(copy);
6  copy2 = [...people2];
7  console.log(copy2);
8
```

copia

## for-of: iterando sobre índices

```
1  people = ["Arle", "Juliana", "valentina", "Ana"];
2  for (const element of people) {
3      console.log(element);
4  }
5
6  people2 = ["Jhon", "Lian"];
7  for (const element of people2.keys()) {
8      console.log(element);
9  }
10
11 for (const [i, e] of people2.entries()) {
12     console.log(i, e);
13 }
```

for-of: iterando sobre [índice, elemento] pares

```
1  const cities = ["Armenia", "pereira", "Cali"];
2  console.log(cities.shift()); // Armenia
3  console.log(cities); // [ 'pereira', 'Cali' ]
4
5  const cities2 = ["Manizales", "Popayan", "Medellin"];
6  console.log(cities2.pop()); // Medellin
7
8  const cities3 = [...cities, ...cities2];
9  console.log(cities3); //[ 'pereira', 'Cali', 'Manizales', 'Popayan' ]
10
11 cities3.splice( start: 1, deleteCount: 2);
12 console.log(cities3);//[ 'pereira', 'Popayan' ]
```

Remueve el primero elemento

Remueve el ultimo elemento

Remueve varios elementos

La programación funcional en JavaScript se enfoca en la manipulación de los datos sin cambiar el estado original de los mismos

```
1 //Map
2 //El método map() devuelve un nuevo array con los resultados de la llamada a una función
3 // proporcionada para cada elemento del array original.
4 const numbers = [1, 2, 3, 4, 5];
5 const squaredNumbers = numbers.map(num => num * num);
6 console.log(squaredNumbers); // [1, 4, 9, 16, 25]
```

1

```
1 //Filter:
2 //El método filter() crea un nuevo array con todos los elementos que pasan
3 // la prueba implementada por la función proporcionada.
4
5 const numbers = [1, 2, 3, 4, 5];
6 const evenNumbers = numbers.filter(num => num % 2 === 0);
7 console.log(evenNumbers); // [2, 4]
```

2

```
1 //Reduce:
2 //El método reduce() ejecuta una función reductora sobre cada elemento del array,
3 // devolviendo un único valor reducido.
4
5 const numbers = [1, 2, 3, 4, 5];
6 const sumOfNumbers = numbers.reduce((acc : number, curr : number) => acc + curr);
7 console.log(sumOfNumbers); // 15
```

3

```
1 //forEach
2 //El método forEach() ejecuta una función para cada elemento del array.
3
4 const numbers = [1, 2, 3, 4, 5];
5 numbers.forEach(num => console.log(num * 2)); // Output: 2 4 6 8 10
```

4

```
1 //Every:
2 //El método every() comprueba si todos los elementos del array pasan la prueba
3 // implementada por la función proporcionada.
4
5 const numbers = [1, 2, 3, 4, 5];
6 const allNumbersAreGreaterThanZero = numbers.every(num => num > 0);
7 console.log(allNumbersAreGreaterThanZero); // true
```

5

```
1 //Some:
2 //El método some() comprueba si al menos un elemento del array pasa
3 // la prueba implementada por la función proporcionada.
4
5 const numbers = [1, 2, 3, 4, 5];
6 const someNumbersAreGreaterThanFour = numbers.some(num => num > 4);
7 console.log(someNumbersAreGreaterThanFour); // true
```

## Métodos: iteración y transformación

```
1 const cities = ["Armenia", "Pereira", "Cali"];
2 1 cities.forEach(x => console.log(x));
3
4 2 console.log(cities.map(x=> x)); // [ 'Armenia', 'Pereira', 'Cali' ]
5
6 const y = cities.find(x => { // Cali
7 3 if(x == 'Cali'){
8     return x;
9 } }
10 );
11 console.log(y);
12
13 const f = cities.filter( x =>{ //[ 'Pereira' ]
14 4 if(x == 'Pereira'){
15     return x;
16 }
17 }
18 );
19 console.log(f);
20
```

## map

Hay tres formas comunes de crear mapas.

```
1 1 const emptyMap = new Map(); // 1. map vacío
2
3 const animals = new Map( entries: [
4     [1, 'horse'],
5     [2, 'dog'],
6     [3, 'gato']
7 ]);
8
9 const categories = new Map()
10     .set(1, 'shoes')
11     .set(2, 'dresses')
12     .set(3, 'stockings');
13
14 const copy = new Map(categories);
15
16 animals.set(4, "giraffe"); // setear un elemento
17 console.log(animals.get(2)); // traer un elemento -> dog
18 console.log(categories.size); // tamaño del map -> 3
19
20 categories.forEach( callbackfn: x => console.log(x));
21 for (const [key, value] of animals) {
22     console.log(key, value);
23 }
24
25 const objAnimals = Object.fromEntries(categories);
26
27 console.log(objAnimals);
```

iterar valores

mostrar clave -  
valor

Convertir mapas a  
objetos

Los métodos y propiedades son:

- `new Map()` – crea el mapa.
- `map.set(clave, valor)` – almacena el valor asociado a la clave.
- `map.get(clave)` – devuelve el valor de la clave. Será `undefined` si la `clave` no existe en map.
- `map.has(clave)` – devuelve `true` si la `clave` existe en map, `false` si no existe.
- `map.delete(clave)` – elimina el valor de la clave.
- `map.clear()` – elimina todo de map.
- `map.size` – tamaño, devuelve la cantidad actual de elementos.

```
1 // Agregar una entrada al mapa
2 const map = new Map();
3 map.set("clave1", "John Doe"); 1
4 map.set("clave2", "John Doe");
5
6 // Obtener el valor de una clave 2
7 const value = map.get("clave1");
8 console.log(value); // John Doe
9
10 // Comprobar si una clave existe 3
11 const exists = map.has("name");
12 console.log(exists); // false
13
14 // Eliminar una entrada 4
15 map.delete("clave1");
16
17 // Obtener todas las claves 5
18 const keys = map.keys();
19 console.log(keys); // ['name']
20
21 // Obtener todos los valores 6
22 const values = map.values();
23 console.log(values); // ['John Doe']
24
25 // Obtener todos los pares de clave-valor 7
26 const entries = map.entries();
27 console.log(entries); // [['name', 'John Doe']]
28
29 // Eliminar todas las entradas 8
30 map.clear();
```

## Set

Un `Set` es una colección de tipo especial: "conjunto de valores" (sin claves), donde cada valor puede aparecer solo una vez.

Sus principales métodos son:

- `new Set(iterable)` – crea el set. El argumento opcional es un objeto iterable (generalmente un array) con valores para inicializarlo.
- `set.add(valor)` – agrega un valor, y devuelve el set en sí.
- `set.delete(valor)` – elimina el valor, y devuelve `true` si el `valor` existía al momento de la llamada; si no, devuelve `false`.
- `set.has(valor)` – devuelve `true` si el valor existe en el set, si no, devuelve `false`.
- `set.clear()` – elimina todo el contenido del set.
- `set.size` – es la cantidad de elementos.

La característica principal es que llamadas repetidas de `set.add(valor)` con el mismo valor no hacen nada. Esa es la razón por la cual cada valor aparece en `Set` solo una vez.

Por ejemplo, vienen visitantes y queremos recordarlos a todos. Pero las visitas repetidas no deberían llevar a duplicados. Un visitante debe ser "contado" solo una vez.

```
1  let set = new Set();
2  const set2 = new Set( values: ['red', 'green', 'blue']);
3  const set3 = new Set()
4      .add('red')
5      .add('green')
6      .add('blue');
7
8  let john = { name: "John" };
9  let pete = { name: "Pete" };
10 let mary = { name: "Mary" };
11
12 set.add(john);
13 set.add(pete);
14 set.add(mary);
15 set.add(john);
16 set.add(mary);
17
18 console.log( set.size ); // 3
19
20 for (let user of set) {
21     console.log(user.name); // John (luego Pete y Mary)
```



```
1  const mySet = new Set([1, 2, 3, 4, 5]); 1
2
3  //Podemos eliminar elementos de un conjunto usando el método delete():
4  mySet.delete(3); 2
5
6  //Podemos verificar si un valor existe en un conjunto usando el método has():
7  console.log(mySet.has(4)); // true 3
8
9  //Podemos iterar sobre los elementos de un conjunto usando un bucle for-of:
10 for (const value of mySet) { 4
11     console.log(value);
12 }
13 //Podemos iterar
14 mySet.forEach(x ⇒ console.log(x)) 5
```

## Matrices

Las matrices de dos dimensiones son un tipo de matriz que tiene dos dimensiones, es decir, se puede visualizar como una tabla de datos. Las matrices de dos dimensiones se pueden usar para almacenar datos de una manera organizada y eficiente.

```
1  let myMatrix = [[1, 2, 3], [4, 5, 6]]; 1
2
3  let firstElement = myMatrix[0][2]; 2
4  console.log(firstElement) // 3
5
6  myMatrix.push([2,8,9]) 3
7
8  // Iterar sobre la matriz utilizando una función
9  myMatrix.forEach((element, rowIndex, columnIndex) ⇒ { 4
10     console.log(` f${rowIndex}: ${element}`);
11 });
```



**Ejercicio 1:** Crear un array de nombres y recorrerlo

- Crea un array con al menos 5 nombres de personas.
- Recorre el array e imprime cada nombre en la consola.
- Imprime la longitud del array.

**Ejercicio 2:** Agregar y eliminar elementos de un array

- Crea un array vacío.
- Agrega al menos 3 elementos al array utilizando diferentes métodos (por ejemplo, push, unshift).
- Elimina el último elemento del array.
- Elimina un elemento específico del array por su índice.
- Imprime el array final después de las modificaciones.

**Ejercicio 3:** Buscar un elemento en un array

- Crea un array con al menos 10 elementos de diferentes tipos (números, strings, booleanos).
- Busca un elemento específico en el array por su valor.
- Imprime la posición (índice) del elemento encontrado o un mensaje si no se encuentra.
- Bonus: Usa diferentes métodos para realizar la búsqueda (por ejemplo, indexOf, find).

**Ejercicio 4:** Ordenar un array

- Crea un array con números aleatorios.
- Ordena el array de forma ascendente.
- Ordena el array de forma descendente.
- Imprime el array antes y después de ordenarlo.

**Ejercicio 5:** Filtrar elementos de un array

- Crea un array con nombres de personas y sus edades.
- Filtra el array para obtener solo los nombres de las personas mayores de edad (por ejemplo, usando filter).
- Imprime los nombres de las personas que cumplen con el criterio.
- Usa un bucle para recorrer el array filtrado y realizar alguna acción con cada elemento (por ejemplo, mostrar un mensaje).

**Ejercicio 6:** Combinar dos arrays

- Crea dos arrays con diferentes elementos.
- Combina los dos arrays en un solo array nuevo.
- Mantén el orden original de los elementos en el array combinado.

Profesor: Arle Morales Ortiz

**Ejercicio 7:** Eliminar elementos duplicados de un array

- Crea un array con elementos duplicados.
- Elimina los elementos duplicados del array.

**Ejercicio 8:** Encontrar el máximo y el mínimo de un array

- Crea un array con números.
- Encuentra el valor máximo y mínimo del array.
- Imprime el valor máximo y mínimo.

**Ejercicio 9:** Agrupar elementos de un array por una propiedad

- Crea un array con objetos que tengan una propiedad en común (por ejemplo, color, tamaño, tipo).
- Agrupa los elementos del array por el valor de la propiedad común.

**Ejercicio 10:** Convertir un array en un objeto

- Crea un array con elementos de diferentes tipos.
- Convierte el array en un objeto, donde cada elemento del array sea una propiedad del objeto.

**Ejercicio 11:**

Un pequeño negocio necesita un sistema para gestionar su inventario de productos. El sistema debe permitir al usuario:

- Agregar nuevos productos: registrar nombre, descripción, precio, cantidad en stock y categoría.(crear array de objetos)
- Listar todos los productos: mostrar información completa de cada producto.
- Buscar un producto por nombre: mostrar información del producto si existe.
- Eliminar un producto: eliminar producto del inventario.
- Actualizar la cantidad en stock: modificar la cantidad disponible de un producto.