



Arrays

clase n° 7

*REC



recordá
poner a grabar la clase

arrays avanzados

Un array es una lista de datos que se almacenan en una sola variable. Los elementos pueden ser de cualquier tipo: números, cadenas de texto, booleanos, incluso otros arrays u objetos.

Los arrays tienen índices (numeración), que comienzan en 0.



```
1 const frutas = ['manzana', 'plátano', 'naranja'];  
2 console.log(frutas[0]); // 'manzana' (primer elemento)  
3 console.log(frutas[2]); // 'naranja' (tercer elemento)
```

métodos básicos de arrays en JavaScript

Los métodos básicos de arrays son herramientas esenciales que nos permiten **agregar, eliminar y manipular** los elementos de un array de manera sencilla.

Los más frecuentes son: **push, pop, shift, unshift.**

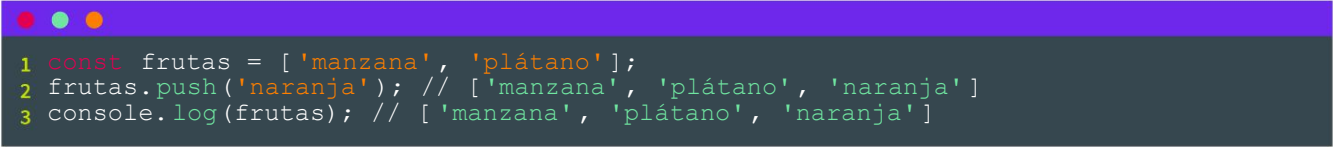
método	acción	lugar del array
push	agrega elementos	al final
pop	elimina el último elemento	al final
shift	elimina el primer elemento	al principio
unshift	agrega elementos	al principio

método push()

push() es un método que agrega uno o más elementos al final de un array.

Después de agregar los elementos, devuelve el nuevo tamaño del array.

Imagina que un array es como una fila de objetos, y **push()** coloca un nuevo objeto **al final de la fila**. Los elementos ya existentes no se mueven, simplemente se añade el nuevo al final.



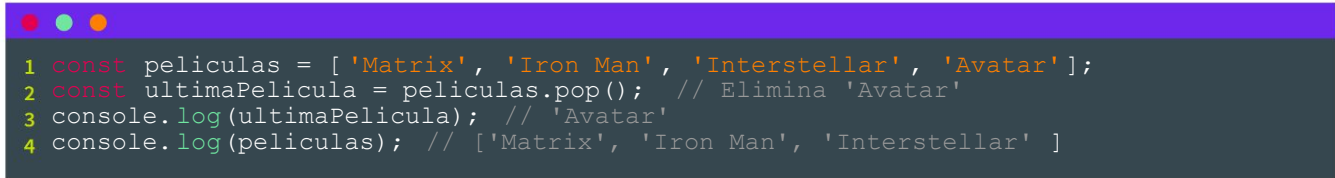
```
1 const frutas = ['manzana', 'plátano'];  
2 frutas.push('naranja'); // ['manzana', 'plátano', 'naranja']  
3 console.log(frutas); // ['manzana', 'plátano', 'naranja']
```

método pop()

pop() elimina el último elemento de un array.
Después de eliminarlo, devuelve el elemento eliminado.

Continuemos imaginando la fila de objetos, **pop()** tomará el **último objeto de la fila, lo elimina y lo entrega**. Los demás elementos del array permanecen igual.

Veámoslo en un ejemplo:



```
1 const peliculas = ['Matrix', 'Iron Man', 'Interstellar', 'Avatar'];  
2 const ultimaPelicula = peliculas.pop(); // Elimina 'Avatar'  
3 console.log(ultimaPelicula); // 'Avatar'  
4 console.log(peliculas); // ['Matrix', 'Iron Man', 'Interstellar']
```

A tener en cuenta: Este método siempre devuelve el último elemento eliminado del array.

método shift()

shift() elimina el primer elemento de un array.

Después de **eliminarlo, devuelve el elemento eliminado y reorganiza los índices** de los elementos restantes.

Imagina que un array es como una fila de personas, y **shift()** saca a la primera **persona de la fila**, mientras los demás se mueven un lugar hacia adelante.

```
1 const filaPersonas = ['Carlos', 'Ana', 'Luis', 'María'];
2 const primeraPersona = filaPersonas.shift(); // Elimina 'Carlos'
3 console.log(primeraPersona); // 'Carlos'
4 console.log(filaPersonas); // ['Ana', 'Luis', 'María']
```

método unshift()

unshift() agrega uno o más elementos al inicio de un array y devuelve la nueva longitud del array.

Continuemos con la idea de la fila de personas, pero ahora agregamos personas **al principio de la fila**. Los demás elementos (o personas) se desplazan hacia la derecha para dejar espacio.

Ejemplo:

```
1 const filaPersonas = ['Ana', 'Luis', 'María'];
2 filaPersonas.unshift('Carlos'); // Agrega 'Carlos' al inicio
3 console.log(filaPersonas); // ['Carlos', 'Ana', 'Luis', 'María']
4
5 filaPersonas.unshift('Sofía', 'Tomás'); // Agrega varios elementos
6 console.log(filaPersonas); // ['Sofía', 'Tomás', 'Carlos', 'Ana', 'Luis', 'María']
```


métodos avanzados de arrays

A medida que los programadores avanzan, se enfrentan a tareas que requieren manipular, transformar o filtrar estos arrays de formas más sofisticadas. Es aquí donde entran en juego los métodos avanzados de arrays. Estos métodos permiten hacer cosas increíbles, como:

- Transformar los elementos de un array.
- Filtrar solo aquellos elementos que cumplen con ciertas condiciones.
- Buscar elementos específicos dentro de un array de manera rápida y sencilla.

método	acción	lugar del array
map	crea un nuevo array con los resultados de aplicar una función a cada elemento	no cambia el array original (devuelve un nuevo array)
filter	crea un nuevo array con los elementos que pasan una prueba o condición	no cambia el array original (devuelve un nuevo array)
find	devuelve el primer elemento que cumple con una condición (o undefined)	solo devuelve un elemento o undefined (no un nuevo array)

método map()

map() es uno de los métodos más versátiles para trabajar con arrays en JavaScript. Se utiliza para crear un nuevo array basado en los elementos de un array existente, pero transformados según una función que se pasa como argumento.

Sintaxis básica:



```
1 let nuevoArray = array.map(funcion);
```

Toma como argumento una función de devolución de llamada (también llamada callback) que se ejecutará en cada elemento del array original.

La función de devolución de llamada puede tener tres parámetros:

- **Elemento actual:** el elemento del array en el que se está trabajando en ese momento.
- **Índice:** la posición del elemento dentro del array.
- **Array original:** el array sobre el que se está aplicando el método.
- **Array original:** el array sobre el que se está aplicando el método.

map() devuelve un nuevo array con los resultados de la ejecución de la función sobre cada elemento, sin modificar el array original.

Ejemplo de uso básico:

Supongamos que tenemos un array de números y queremos crear un nuevo array donde cada número esté multiplicado por 2.

```
1 let numeros = [1, 2, 3, 4];  
2 let dobles = numeros.map(function(num) {  
3   return num * 2;  
4 });  
5 console.log(dobles); // [2, 4, 6, 8]
```

método find()

find() se utiliza para buscar un elemento específico dentro de un array. Devuelve el primer elemento que cumpla con la condición especificada en una función. Si ningún elemento cumple la condición, devuelve **undefined**.

```
1 let elementoEncontrado = array.find(function);
```

Toma una función de devolución de llamada (callback) como argumento. Esta función de devolución de llamada se ejecuta en cada elemento del array hasta que encuentra uno que cumpla con la condición. Una vez que encuentra el primer elemento que satisface la condición, detiene la búsqueda y devuelve ese elemento. Si ningún elemento cumple la condición, **find** devuelve **undefined**. Supongamos que tenemos un array de números y queremos encontrar el primer número mayor que 10.

```
1 let numeros = [5, 8, 12, 20, 7];  
2 let encontrado = numeros.find(function (num) {  
3   return num > 10;  
4 });  
5 console.log(encontrado); // 12
```

método filter()

filter() se utiliza para crear un nuevo array que contiene sólo los elementos del array original que cumplen con una condición específica. A diferencia de **find()**, que devuelve un único elemento, **filter()** devuelve todos los elementos que cumplan con la condición. Sintaxis básica:

```
1 let nuevoArray = array.filter(function);
```

Recorre cada elemento del array original. Para cada elemento, ejecuta una función de devolución de llamada (callback). Si la función de devolución de llamada devuelve **true** para un elemento, ese elemento se incluye en el nuevo array. Si devuelve **false**, el elemento se omite. El array original no se modifica. Supongamos que tienes un array de números y quieres obtener un nuevo array que contenga solo los números mayores que 10.

```
1 let numeros = [5, 8, 12, 20, 7];  
2 let mayoresDeDiez = numeros.filter(function(num) {  
3   return num > 10;  
4 });  
5 console.log(mayoresDeDiez); // [12, 20]
```



revolución*
digital_