Métodos de Arrays en JavaScript

1. every()

Verifica si todos los elementos cumplen una condición.
- Sintaxis: array.every(callback(elemento[, índice[, array]])[, thisArg])
- Retorna: true` si todos los elementos cumplen la condición; `false` en caso contrario.
- No modifica el array original.

```
const numbers = [2, 4, 6, 8];
const allEven = numbers.every(num => num % 2 === 0); // true
```

2. filter()

Crea un nuevo array con elementos que cumplen una condición.
- Sintaxis: array.filter(callback(elemento[, índice[, array]])[, thisArg])
- Retorna: Un nuevo array con los elementos filtrados.
- No modifica el array original

```
const numbers = [1, 2, 3, 4];
const evenNumbers = numbers.filter(num => num % 2 === 0); // [2, 4]
```

3. find()

Retorna el primer elemento que cumple una condición.
- Sintaxis: array.find(callback(elemento[, índice[, array]])[, thisArg])`
- Retorna: El elemento encontrado o `undefined`.
- No modifica el array original.

```
const numbers = [5, 12, 8, 130];
const found = numbers.find(num => num > 10); // 12
```

4.findIndex()

Retorna el índice del primer elemento que cumple una condición.
-Sintaxis: array.findIndex(callback(elemento[, índice[, array]])[, thisArg])
-Retorna: El índice del elemento encontrado o -1.
- No modifica el array original.

```
const numbers = [5, 12, 8, 130];
const index = numbers.findIndex(num => num > 10); // 1
```

5. forEach()

Ejecuta una función para cada elemento del array.
- Sintaxis:array.forEach(callback(elemento[, índice[, array]])[, thisArg])
- Retorna: undefined.
- No modifica el array original (a menos que la callback lo haga).

```
const numbers = [1, 2, 3];
numbers.forEach(num => console.log(num)); // 1, 2, 3
```

6.includes()

Verifica si el array contiene un elemento.
- Sintaxis: array.includes(valorBuscado[, índiceDesde])
- Retorna: `true` o `false`.
-No modifica el array original.

```
const fruits = ['apple', 'banana'];
const hasApple = fruits.includes('apple'); // true
```

7. indexOf()

Retorna el primer índice donde se encuentra un elemento.
-Sintaxis: array.indexOf(valorBuscado[, índiceDesde])
-Retorna: El índice del elemento o -1.
- No modifica el array original.

```
const fruits = ['apple', 'banana'];
const index = fruits.indexOf('banana'); // 1
```

8. join()

Une los elementos del array en un string.
- Sintaxis: `array.join([separador])`
- Retorna: Un string con los elementos unidos.
- No modifica el array original.

```
const elements = ['Fire', 'Air', 'Water'];
const joined = elements.join(' - '); // "Fire - Air - Water"
```

9.map()

Crea un nuevo array transformando cada elemento.
- Sintaxis: `array.map(callback(elemento[, índice[, array]])[, thisArg])`
- Retorna: Un nuevo array con los elementos transformados.
- No modifica el array original.

```
const numbers = [1, 2, 3];
const doubled = numbers.map(num => num * 2); // [2, 4, 6]
```

10.pop()

Elimina el último elemento del array.
- Sintaxis: `array.pop()`
- Retorna: El elemento eliminado.
- Modifica el array original.

```
const fruits = ['apple', 'banana'];
const last = fruits.pop(); // 'banana' → fruits = ['apple']
```

11.push()

Agrega elementos al final del array.
- Sintaxis: `array.push(elemento1[, elemento2[, ...[, elementoN]]])`
- Retorna: La nueva longitud del array.
- Modifica el array original.

```
const fruits = ['apple'];
fruits.push('banana'); // 2 → fruits = ['apple', 'banana']
```

12. reduce()

Reduce el array a un único valor aplicando una función acumuladora.
- Sintaxis: `array.reduce(callback(acumulador, elemento[, índice[, array]])[, valorInicial])`
- Retorna: El valor acumulado.
- No modifica el array original.

```
const numbers = [1, 2, 3];
const sum = numbers.reduce((acc, num) => acc + num, 0); // 6
```

13. reverse()

Invierte el orden de los elementos del array.
- Sintaxis: `array.reverse()`
- Retorna: El array invertido.
- Modifica el array original.

```
const numbers = [1, 2, 3];
numbers.reverse(); // [3, 2, 1]
```

14. shift()

Elimina el primer elemento del array.
- Sintaxis: `array.shift()`
- Retorna: El elemento eliminado.
- Modifica el array original.

```
const fruits = ['apple', 'banana'];
const first = fruits.shift(); // 'apple' → fruits = ['banana']
```

15.slice()

Crea una copia superficial de una porción del array.
- Sintaxis: `array.slice([índiceInicio[, índiceFin]])`
- Retorna: Un nuevo array con los elementos seleccionados.
- No modifica el array original.

```
const numbers = [1, 2, 3, 4];
const subArray = numbers.slice(1, 3); // [2, 3]
```

16.some()

Verifica si al menos un elemento cumple una condición.
- Sintaxis: `array.some(callback(elemento[, índice[, array]])[, thisArg])`
- Retorna: `true` o `false`.
- No modifica el array original.

```
const numbers = [1, 2, 3];
const hasEven = numbers.some(num => num % 2 === 0); // true
```

17.sort()

Ordena los elementos del array.
- Sintaxis: `array.sort([funcionDeComparacion])`
- Retorna: El array ordenado.
- Modifica el array original.

```
const numbers = [3, 1, 4];
numbers.sort((a, b) => a - b); // [1, 3, 4]
```

18.split()

Método de String: Divide un string en un array de substrings.
- Sintaxis: `string.split([separador[, limite]])`
- Retorna: Un array de strings.

```
const text = 'Hola Mundo';
const words = text.split(' '); // ['Hola', 'Mundo']
```

19.unshift()

Agrega elementos al inicio del array.
- Sintaxis: `array.unshift(elemento1[, elemento2[, ...[, elementoN]]])`
- Retorna: La nueva longitud del array.
- Modifica el array original.

```
const fruits = ['banana'];
fruits.unshift('apple'); // 2 → fruits = ['apple', 'banana']
```