

JAVASCRIPT

Arrays

The logo consists of the letters 'JS' in a bold, dark blue, sans-serif font. The 'J' and 'S' are connected. This logo is centered within a light yellow square, which is itself centered on a larger yellow background.

JS

¿Qué es un Array?

Son **colecciones** de datos en **formato de matriz** o vector.

Nos permiten **agrupar conjuntos de valores** relacionados en una **misma variable**.

Estructura del Array

Los **arreglos** poseen un **formato de lista**, es decir, una secuencia de valores agrupados dentro de un par de **[]** y separados por **coma**.

```
const frutas = ['Manzana', 'Pera', 'Frutilla', 'Kiwi', 'Sandía'];
```

Estos **valores** ocupan una posición dentro del array. A esa posición se la conoce como **índice** y **siempre comienza en 0**.

En el ejemplo anterior, al tener **5 valores**, estos comienzan desde el índice **0** (Manzana) hasta el índice **4** (Sandía).

Acceso a los datos

Para **acceder** a un valor del array podemos hacerlo **conociendo el índice** del mismo de la siguiente manera:

```
const frutas = ['Manzana', 'Pera', 'Frutilla', 'Kiwi', 'Sandía'];  
console.log(frutas[2]); // Frutilla
```

Otra forma, a partir de la especificación de EcmaScript 2022 es **con el método .at()**:

```
console.log(frutas.at(1)); // Pera
```

Tipos y cantidad

Estas **estructuras** son **dinámicas** y además aceptan distintos tipos de datos al mismo tiempo.

```
const datos = ["José", 23, true, "Calle Falsa 123"];
```

Por otra parte, podemos acceder a la **propiedad length** del array para conocer la cantidad de valores que contiene.

```
console.log(datos.length); // 4
```

A diferencia de otros lenguajes, en **javascript**, un array no tiene que tener definido la cantidad de elementos a guardar antes de ser creado.

Array Methods

Son **funciones nativas** que poseen los arrays y que nos **permiten** trabajar con ellos de forma sencilla, poniendo a disposición herramientas de **adición, eliminación, filtrado y ordenado** de valores entre otras cosas.

Añadir o eliminar elementos

Estos métodos cambian el arreglo, por lo que cada vez que los utilicemos **estaremos modificando los valores original** del array en cuestión.

push(): agrega un valor al **final** y retorna el nuevo length.

unshift(): agrega un valor al **principio** y retorna el nuevo length.

pop(): elimina el **último** valor y lo retorna.

shift(): elimina el **primer** valor y lo retorna.

```
frutas.push('Ananá'); // Agregamos un elemento al final del array
frutas.unshift('Melón'); // Agregamos un elemento al inicio del array

console.log(frutas);

// ['Melón', 'Manzana', 'Pera', 'Frutilla', 'Kiwi', 'Sandía', 'Ananá'];

frutas.pop(); // Elimina el último elemento del array
frutas.shift(); // Elimina el primer elemento del array

console.log(frutas);

// ['Manzana', 'Pera', 'Frutilla', 'Kiwi', 'Sandía'];
```

Concatenar elementos

Nos permiten **combinar** arrays o crear cadenas de texto a partir de **los valores de un mismo arreglo**:

concat(): combina 2 o más arrays pasados por parámetro.

```
const precioRemeras = [100, 320, 257];
const precioMedias = [50, 35, 23];
const precios = precioRemeras.concat(precioMedias);

console.log(precios);

// [100, 320, 257, 50, 35, 23]
```

join(): crea una cadena de texto a partir de todos **los valores de un array**. Recibe por **parámetro** un **separador de elementos** de forma opcional.

```
frutas.join('-')

console.log(frutas);

'Manzana - Pera - Frutilla - Kiwi - Sandía'
```


Separar o Cortar

split(): creamos un array a partir de una cadena de texto.

El parámetro es la condición que separa a los elementos en la cadena.

slice(): devuelve una porción del array desde un rango definido.

Por parámetro pasaremos la posición inicial y final de los elementos a cortar.

```
const nombres = 'Julieta, Carlos, Pedro, Juana';  
nombres = nombres.split(',');  
console.log(nombres);  
  
// ['Julieta', 'Carlos', 'Pedro', 'Juana']
```

```
const animales = ['Pato', 'Perro', 'Gato', 'Loro', 'Puma'];  
console.log(animales.slice(2, 4));  
  
// ['Gato', 'Loro']
```

Ordenar

`sort()`: nos permite **ordenar alfabéticamente** los elementos de un array.

```
const animales = ['Pato', 'Perro', 'Gato', 'Loro', 'Puma'];  
  
console.log(animales.sort());  
  
// ['Gato', 'Loro', 'Pato', 'Perro', 'Puma']
```

En caso que deseemos **ordenar números**, el **método sort resulta errático** ya que valores como **10 y 100, los tomaría consecutivos** ya que el 1 siempre viene antes del 2 alfabéticamente.

Para evitar esto, podemos aplicar el siguiente hack:

```
const precios = [95, 5, 25, 10, 250];  
  
precios.sort((a, b) => a - b);  
// [5, 10, 25, 95, 250]
```

Array Functions

Son **métodos de array** que originalmente **reciben** una función de **callback** por parámetro para obtener cierto resultado.

Iterador

`forEach()`: este método no retorna ningún valor, sino que solo se limita a **ejecutar el callback** que le pasemos por cada elemento del array.

```
/**
 * foreach
 */

const arr = [1, 2, 3, 4, 5, 6];

arr.forEach(item => {
  console.log(item); // output: 1 2 3 4 5 6
});
```

Validadores

`every()`: verifica si **todos los elementos** en el arreglo **pasan la prueba** implementada por la función dada.

`some()`: verifica si al menos **uno de los elementos** en el arreglo **pasan la prueba** implementada por la función dada.

```
/**
 * every
 */

const arr = [1, 2, 3, 4, 5, 6];

// all elements are greater than 4
const greaterFour = arr.every(num => num > 4);
console.log(greaterFour); // output: false

// all elements are less than 10
const lessTen = arr.every(num => num < 10);
console.log(lessTen); // output: true
```

```
/**
 * some
 */

const arr = [1, 2, 3, 4, 5, 6];

// at least one element is greater than 4?
const largeNum = arr.some(num => num > 4);
console.log(largeNum); // output: true

// at least one element is less than or equal to 0?
const smallNum = arr.some(num => num <= 0);
console.log(smallNum); // output: false
```

Reducer

`reduce()`: se ejecuta **por cada elemento** del array y va acumulando en una variable el valor anterior sumando el valor actual de esa iteración.

```
/**
 * reduce
 */

const arr = [1, 2, 3, 4, 5, 6];

const sum = arr.reduce((total, value) => total + value, 0);
console.log(sum); // 21
```

Transformador

`map()`: crea un **nuevo arreglo** con el **resultado de la función de callback** pasada por parámetro.

```
/**
 * map
 */

const arr = [1, 2, 3, 4, 5, 6];

// add one to every element
const oneAdded = arr.map(num => num + 1);
console.log(oneAdded); // output [2, 3, 4, 5, 6, 7]

console.log(arr); // output: [1, 2, 3, 4, 5, 6]
```

Filtro

`filter()`: **filtra los elementos que cumplen cierta condición**. Podríamos usarlo cuando tenemos un array y necesitamos filtrar datos.

```
/**
 * filter
 */


const arr = [1, 2, 3, 4, 5, 6];

// item(s) greater than 3
const filtered = arr.filter(num => num > 3);
console.log(filtered); // output: [4, 5, 6]

console.log(arr); // output: [1, 2, 3, 4, 5, 6]
```


Buscador

`find()`: busca en un arreglo según una condición y **devuelve** el *primer valor que logre cumplirla*.



```
const precios = [200, 40, 1000];  
precios.find((precio) => precio > 350); // 1000
```

Objetos y Arrays

Objetos y Arrays

Estas estructuras **se llevan muy bien** y es muy común **utilizarlas en conjunto** para **crear colecciones de datos robustas** que nos permitan contar con estructuras fácilmente iterables.

En este caso **tenemos un array** con una colección de tareas donde **cada tarea** es **un objeto** con sus **respectivas propiedades**.

¿Cómo podríamos recorrer esta estructura para obtener sólo los nombres de cada tarea? 🤔

```
let tasks = [  
  {  
    id: 1,  
    day: 'Lunes',  
    task: "Leer un libro",  
    state: "Pendiente"  
  },  
  {  
    id: 2,  
    day: 'Miércoles',  
    task: "Sacar al Perro",  
    state: "Pendiente"  
  },  
  {  
    id: 3,  
    day: 'Viernes',  
    task: "Jugar Videojuegos",  
    state: "Pendiente"  
  },  
];
```