El bucle forEach en JavaScript es un método del objeto Array que permite iterar sobre cada elemento de un array y ejecutar una función en cada uno de ellos.

**Sintaxis**

array.forEach(function(elemento, indice, array) {

// Código a ejecutar en cada iteración

});

O con una función de flecha:

array.forEach((elemento, indice, array) => {

// Código a ejecutar en cada iteración

});

**Parámetros:**

1. **elemento** → El valor del elemento actual en la iteración.
2. **indice** *(opcional)* → La posición del elemento en el array.
3. **array** *(opcional)* → El array sobre el que se está iterando.

**Ejemplo básico:**

let numeros = [1, 2, 3, 4];

numeros.forEach(num => {

console.log(num \* 2);

});

// Salida: 2, 4, 6, 8

**Ejemplo con índice:**

let frutas = ["Manzana", "Banana", "Naranja"];

frutas.forEach((fruta, indice) => {

console.log(`Índice ${indice}: ${fruta}`);

});

/\* Salida:

Índice 0: Manzana

Índice 1: Banana

Índice 2: Naranja

\*/

**Diferencias con for y map**

* **No se puede usar break ni continue**, ya que forEach no puede ser interrumpido.
* **No devuelve un nuevo array**, a diferencia de map(), que sí lo hace.

Ejemplo comparativo con map:

let duplicados = numeros.map(num => num \* 2);

console.log(duplicados); // [2, 4, 6, 8]

En forEach, si quisiéramos modificar el array, tendríamos que hacerlo manualmente.

**Casos en los que forEach NO es ideal**

1. **Cuando necesitas detener la ejecución** → Mejor usar for o for...of.
2. **Si necesitas devolver un nuevo array** → Mejor usar map().
3. **Si trabajas con async/await** → forEach no espera promesas, en su lugar usa for...of.

Ejemplo incorrecto con async/await:

async function procesar() {

let datos = [1, 2, 3];

datos.forEach(async (num) => {

let resultado = await fetch(`https://api.com/data/${num}`);

console.log(await resultado.json());

});

}

Este código **NO** espera que las promesas se resuelvan en orden. Usa for...of en su lugar.

**Conclusión**

forEach es útil para recorrer arrays de manera sencilla cuando no necesitas modificar el array original ni interrumpir la ejecución. Sin embargo, en ciertos casos, for, map, o for...of pueden ser mejores opciones.

**for...of en JavaScript**

El bucle for...of se usa para recorrer iterables (como arrays, strings, mapas, sets, etc.) de manera sencilla y eficiente.

**Sintaxis**

for (let variable of iterable) {

// Código a ejecutar en cada iteración

}

* **variable** → Almacena el valor del elemento en cada iteración.
* **iterable** → El objeto sobre el cual iteramos (array, string, etc.).

**Ejemplo con Arrays**

let numeros = [10, 20, 30, 40];

for (let numero of numeros) {

console.log(numero);

}

/\* Salida:

10

20

30

40

\*/

Aquí, numero toma el valor de cada elemento del array en cada iteración.

**Ejemplo con Strings**

Se puede recorrer cada carácter de una cadena:

let palabra = "Hola";

for (let letra of palabra) {

console.log(letra);

}

/\* Salida:

H

o

l

a

\*/

**Ejemplo con Set (conjuntos sin valores repetidos)**

let conjunto = new Set(["🍎", "🍌", "🍇"]);

for (let fruta of conjunto) {

console.log(fruta);

}

/\* Salida:

🍎

🍌

🍇

\*/

**Ejemplo con Map (objetos clave-valor iterables)**

Los Map permiten recorrer tanto **clave** como **valor**:

let productos = new Map([

["manzana", 3],

["banana", 5],

["naranja", 2]

]);

for (let [fruta, cantidad] of productos) {

console.log(`Hay ${cantidad} ${fruta}(s).`);

}

/\* Salida:

Hay 3 manzana(s).

Hay 5 banana(s).

Hay 2 naranja(s).

\*/

**Diferencia entre for...of y for...in**

| **Característica** | **for...of** | **for...in** |
| --- | --- | --- |
| Itera sobre | Valores de un iterable | Índices o claves de un objeto |
| Uso común | Arrays, Strings, Sets, Maps | Objetos ({}) |
| Ejemplo | for (let valor of array) | for (let key in objeto) |

Ejemplo comparativo:

let array = ["a", "b", "c"];

for (let valor of array) {

console.log(valor); // a, b, c

}

for (let indice in array) {

console.log(indice); // 0, 1, 2

}

**Cuándo usar for...of**

✅ Cuando necesitas recorrer **valores** en un iterable.  
✅ Es más limpio y fácil de leer que forEach en algunas situaciones.  
✅ Funciona mejor con async/await que forEach.

🚫 **No usar en objetos normales {}** → Para eso, usa for...in o Object.entries().

**Conclusión**

El bucle for...of es una forma simple y legible de recorrer estructuras de datos iterables en JavaScript. Es ideal para iterar sobre arrays, strings, Maps y Sets sin preocuparte por los índices. 🚀

**for...in en JavaScript**

El bucle for...in se usa para recorrer las **propiedades enumerables** de un objeto. En cada iteración, devuelve la **clave** (nombre de la propiedad) en lugar del valor.

**Sintaxis**

for (let clave in objeto) {

// Código a ejecutar

}

* **clave** → Almacena el nombre de la propiedad en cada iteración.
* **objeto** → El objeto sobre el cual iteramos.

**Ejemplo con Objetos**

let persona = { nombre: "Juan", edad: 30, ciudad: "Madrid" };

for (let clave in persona) {

console.log(`${clave}: ${persona[clave]}`);

}

/\* Salida:

nombre: Juan

edad: 30

ciudad: Madrid

\*/

Aquí, clave toma los nombres de las propiedades (nombre, edad, ciudad), y accedemos a los valores con persona[clave].

**Ejemplo con Arrays (NO recomendado)**

let colores = ["Rojo", "Verde", "Azul"];

for (let indice in colores) {

console.log(`Índice ${indice}: ${colores[indice]}`);

}

/\* Salida:

Índice 0: Rojo

Índice 1: Verde

Índice 2: Azul

\*/

⚠️ Aunque for...in funciona con arrays, **NO es recomendable** porque itera sobre las **claves (índices)** en lugar de los valores.

En su lugar, usa for...of:

for (let color of colores) {

console.log(color);

}

/\* Salida:

Rojo

Verde

Azul

\*/

**Ejemplo con Objetos y Object.entries()**

Para recorrer tanto **clave** como **valor**, usa Object.entries():

let usuario = { nombre: "Ana", edad: 25 };

for (let [clave, valor] of Object.entries(usuario)) {

console.log(`${clave}: ${valor}`);

}

/\* Salida:

nombre: Ana

edad: 25

\*/

**Diferencia entre for...in y for...of**

| **Característica** | **for...in** | **for...of** |
| --- | --- | --- |
| Itera sobre | Claves (propiedades) | Valores |
| Uso común | Objetos {} | Arrays, Strings, Maps, Sets |
| Ejemplo | for (let clave in objeto) | for (let valor of array) |

Ejemplo comparativo:

let array = ["a", "b", "c"];

for (let indice in array) {

console.log(indice); // 0, 1, 2

}

for (let valor of array) {

console.log(valor); // a, b, c

}

**Cuándo usar for...in**

✅ Cuando necesitas recorrer las **propiedades** de un objeto.  
✅ Para depurar claves de objetos dinámicos.

🚫 **No usar en arrays**, porque puede incluir propiedades heredadas y el orden de iteración no está garantizado.

**Conclusión**

El bucle for...in es ideal para recorrer las claves de un objeto, pero no es recomendable para arrays. Si necesitas los valores de un array, usa for...of en su lugar. 🚀

**El método map() en JavaScript**

map() es un método de los arrays en JavaScript que crea un **nuevo array** aplicando una función a cada uno de los elementos del array original.

**Sintaxis**

let nuevoArray = array.map((elemento, indice, array) => {

// Operación con el elemento

return nuevoValor;

});

* **elemento** → Valor del elemento actual.
* **indice** *(opcional)* → Posición del elemento en el array.
* **array** *(opcional)* → El array sobre el cual se ejecuta map().
* Devuelve un **nuevo array**, sin modificar el original.

**Ejemplo básico**

let numeros = [1, 2, 3, 4];

let duplicados = numeros.map(num => num \* 2);

console.log(duplicados); // [2, 4, 6, 8]

console.log(numeros); // [1, 2, 3, 4] (No cambia)

Aquí, map() multiplica cada número por 2 y devuelve un **nuevo array** con los valores transformados.

**Ejemplo con índice**

let nombres = ["Ana", "Juan", "Luis"];

let conIndices = nombres.map((nombre, indice) => `${indice + 1}. ${nombre}`);

console.log(conIndices);

/\* Salida:

["1. Ana", "2. Juan", "3. Luis"]

\*/

**Ejemplo transformando objetos**

let productos = [

{ nombre: "Laptop", precio: 1000 },

{ nombre: "Mouse", precio: 50 },

{ nombre: "Teclado", precio: 80 }

];

let preciosConIVA = productos.map(prod => ({

...prod,

precio: prod.precio \* 1.21 // Agrega 21% de IVA

}));

console.log(preciosConIVA);

/\* Salida:

[

{ nombre: 'Laptop', precio: 1210 },

{ nombre: 'Mouse', precio: 60.5 },

{ nombre: 'Teclado', precio: 96.8 }

]

\*/

map() permite transformar cada objeto sin modificar el original.

**Diferencias entre map() y forEach()**

| **Característica** | **map()** | **forEach()** |
| --- | --- | --- |
| Retorna un nuevo array | ✅ Sí | ❌ No |
| Modifica el array original | ❌ No | ❌ No |
| Se usa para transformar datos | ✅ Sí | ❌ No (solo ejecuta código) |

Ejemplo comparativo:

let numeros = [1, 2, 3, 4];

let conMap = numeros.map(num => num \* 2); // [2, 4, 6, 8]

let conForEach = numeros.forEach(num => num \* 2); // undefined

🚀 **Usa map() si necesitas un nuevo array con los valores transformados.**  
🛑 **Usa forEach() si solo necesitas ejecutar una acción sin modificar datos.**

**Cuándo usar map()**

✅ Cuando necesitas **transformar** los elementos de un array.  
✅ Cuando necesitas **mantener el array original sin modificarlo**.  
🚫 **No usar si no necesitas el nuevo array** → En ese caso, usa forEach().

**Conclusión**

El método map() es una poderosa herramienta para transformar arrays de forma funcional, manteniendo el original intacto y devolviendo un nuevo array con los valores modificados. 🚀