Métodos Mutables e Inmutables en Arrays

📘 1️⃣ Introducción

En JavaScript, los arrays son estructuras dinámicas que pueden cambiar su contenido. Sin embargo, **no todos los métodos de array se comportan igual**:

- Algunos **modifican directamente el array original** → se llaman **mutables**.
- Otros **devuelven un nuevo array** sin tocar el original → se llaman **inmutables**.

👉 Conocer esta diferencia es **fundamental** para evitar errores, especialmente cuando trabajamos con datos compartidos o frameworks modernos como **React**, donde la inmutabilidad es una buena práctica.

🔁 2️⃣ Métodos mutables (modifican el array original)

Los métodos **mutables** cambian el contenido o el orden del array original.
Cuando los usas, **pierdes el estado anterior** del array.

📦 Ejemplos de métodos mutables:

Método	Descripción	Ejemplo
push()	Agrega uno o más elementos al final	arr.push(4)
pop()	Elimina el último elemento	arr.pop()
shift()	Elimina el primer elemento	arr.shift()
unshift()	Agrega elementos al principio	arr.unshift(1)
splice()	Elimina o reemplaza elementos en cualquier posición	arr.splice(1, 1)
sort()	Ordena el array	arr.sort()
reverse()	Invierte el orden del array	arr.reverse()
fill()	Rellena el array con un valor	arr.fill(0)
copyWithin()	Copia elementos dentro del mismo array	arr.copyWithin(1, 0)

💡 Ejemplo práctico (mutable)

```
let frutas = ["manzana", "pera", "uva"];
frutas.splice(1, 1, "naranja"); // elimina 1 desde índice 1 y agrega "naranja"

console.log(frutas);
// ["manzana", "naranja", "uva"]
```

📘 Observa:

El array original frutas fue modificado.
Si lo necesitáramos más tarde con su contenido inicial, ya no lo tenemos.

🧩 3️⃣ Métodos inmutables (no modifican el array original)

Los métodos **inmutables** devuelven un **nuevo array transformado o filtrado**, dejando el original intacto.
Esto facilita mantener el **estado original** y evita efectos secundarios.

📦 Ejemplos de métodos inmutables:

Método	Descripción	Ejemplo
map()	Crea un nuevo array transformando los elementos	arr.map(x => x * 2)
filter()	Crea un nuevo array con los elementos que cumplen la condición	arr.filter(x => x > 10)
slice()	Crea una copia parcial del array	arr.slice(0, 2)
concat()	Une arrays y devuelve uno nuevo	arr.concat(otroArr)
flat()	Aplana arrays anidados	arr.flat()
flatMap()	Combina map() y flat()	arr.flatMap(x => [x, x * 2])
toSorted()	Ordena sin modificar el original (nuevo en ES2023)	arr.toSorted()
toReversed()	Invierte sin modificar el original (nuevo en ES2023)	arr.toReversed()

toSpliced()	Similar a splice(), pero devuelve una copia	arr.toSpliced(1, 2)

💡 Ejemplo práctico (inmutable)

```
let numeros = [10, 20, 30, 40];
let mayores = numeros.filter(n => n > 20);

console.log(mayores); // [30, 40]
console.log(numeros); // [10, 20, 30, 40] (no cambia)
```

📘 **Observa:**
El array original se conserva igual, y filter() devuelve un **nuevo array** con los resultados.

⚙️ 4️⃣ Diferencias visuales entre mutables e inmutables

Tipo	Ejemplo	Modifica el original	Devuelve nuevo array
Mutable	arr.push(4)	✅ Sí	❌ No
Mutable	arr.sort()	✅ Sí	❌ No
Inmutable	arr.filter()	❌ No	✅ Sí
Inmutable	arr.slice()	❌ No	✅ Sí
Inmutable (nuevo ES2023)	arr.toSorted()	❌ No	✅ Sí

🔬 5️⃣ Por qué esto importa

- En **JavaScript tradicional**, puedes usar ambos tipos según necesidad.
- En **React** y entornos modernos, se recomienda usar **métodos inmutables**, porque el cambio de estado debe hacerse **creando nuevas copias** (no mutando los datos directamente).
- La **inmutabilidad** evita errores de referencias, re-renderizados incorrectos y efectos secundarios.

📘 **Ejemplo React-like:**

// ❌ Mal (mutando el estado)

```
estado.push(nuevoElemento);
```

// ✅ Bien (creando un nuevo array)

```
estado = [...estado, nuevoElemento];
```

🧩 6️⃣ **Ejercicio práctico para los alumnos**

🎯 **Objetivo:**

Distinguir entre métodos mutables e inmutables y observar su efecto.

```
let productos = ["teclado", "ratón", "monitor"];

// Mutable
productos.push("auriculares");
console.log("Mutable:", productos);

// Inmutable
let nuevos = productos.concat("impresora");
console.log("Inmutable:", nuevos);
console.log("Original:", productos);
```

Salida:

Mutable: ["teclado", "ratón", "monitor", "auriculares"]

Inmutable: ["teclado", "ratón", "monitor", "auriculares", "impresora"]

Original: ["teclado", "ratón", "monitor", "auriculares"]

📘 Así verán claramente que los mutables alteran el contenido, mientras que los inmutables **crean una nueva versión del array**.

🎯 7️⃣ Conclusión

Concepto	Mutable	Inmutable
Qué hacen	Cambian directamente el array original	Devuelven un nuevo array
Riesgo	Pérdida del estado original	Ninguno
Ideal para	Algoritmos simples o controlados	Programación funcional y React
Ejemplos	push, pop, splice, sort	map, filter, slice, concat