Clase Básica de JavaScript (ES6) — Bases para React

Diego Muñoz

2 de septiembre de 2025

Introducción

- Repaso de JS moderno (ES6+).
- Conceptos clave que React usa a diario.
- Enfoque en funciones anónimas, arrow, inmutabilidad, arrays.

JavaScript: fundamentos

- Corre en navegador y en Node.js.
- Tipado dinámico.
- Primitivos: string, number, boolean, null, undefined, bigint, symbol.
- Objetos, arrays y funciones → por referencia.

Igualdad y coerción

- Evita `==`. Usa `===`.
- Valores falsy: 0, “”, null, undefined, NaN, false.

```
"" == 0      // true
"" === 0     // false
Boolean("false") // true
```

Scope y variables

- `let` y `const` tienen alcance de bloque.
- `const`: no re-asignable (pero objetos mutan propiedades).
- Evitar `var`.

```
{
  let a = 1;
  const b = 2;
}
// fuera del bloque no existen
```

Funciones

- Declaración: `function suma(a,b) {...}`
- Expresión: `const f = function(a,b){...}`
- Arrow: `const f = (a,b) => a+b`

Arrow tiene `this` léxico → clave en React.

Arrow functions: ejemplos

```
const inc = x => x + 1;
const sum = (a,b) => a + b;
const toUser = name => ({ name });
```

Closures (contador)

```
function makeCounter(initial=0) {
  let value = initial;
  return {
    next: () => ++value,
    reset: () => (value = initial)
  };
}
```

- React hooks usan esta idea de “recordar” valores.

Objetos y arrays

- Destructuring para extraer valores.
- Spread (`...`) para copiar/combinar.
- Cambios siempre inmutables.

```
const base = { timeout:1000 };
const override = { timeout:3000 };
const merged = { ...base, ...override };
```

Inmutabilidad

- Evitar mutar arrays/objetos.
- Crear copias nuevas con spread o map.
- React detecta cambios por referencia.

```
const updated = { ...settings, timeout:5000 };
```

Arrays funcionales

- map: transforma → nuevo array.
- filter: filtra por condición.
- reduce: acumula a un valor.

```
const nums = [1,2,3];
nums.map(n => n*2);   // [2,4,6]
nums.filter(n => n%2===0); // [2]
```

Reduce (ejemplo práctico)

```
const users = [
  {id:1, name:"Ana", age:22},
  {id:2, name:"Luis", age:19}
];

const byAge = users.reduce((acc,u) => {
  if (acc[u.age] == null) {
    acc[u.age] = [];
  }
  acc[u.age].push(u.name);
  return acc;
}, {});
```

Cortocircuito lógico

- `cond || valor` → por defecto.
- `cond && expr` → condicional.

```
const title = input || "Sin título";
flag && doSomething();
```

En React: `cond && <Componente />`.

DOM y funciones anónimas

```
<button id="btn">Contar</button>
<output id="out">0</output>
<script>
  let v = 0;
  const out = document.getElementById("out");
  document.getElementById("btn")
    .addEventListener("click", () => {
      out.textContent = String(++v);
    });
</script>
```

Puente hacia React

- Funciones puras → componentes.
- Props ~ parámetros inmutables.
- Estado ~ closures (React lo gestiona).
- Render condicional con `&&` o `?:`.
- Listas con `array.map(...)`.

Patrones clave

1. Evitar mutaciones.
2. Funciones pequeñas y puras.
3. Destructuring en parámetros.
4. Callbacks anónimos en handlers.
5. Keys estables en listas.

Anti-patrones

- Mutar objetos/arrays recibidos.
- Variables globales.
- Funciones con demasiadas responsabilidades.
- Usar `==` en lugar de `===`.

Proyecto de ejemplo

Referir a ejemplo en GitHub.

Resumen

- Funciones anónimas y arrow.
- Closures para entender estado.
- Inmutabilidad como regla.
- map/filter/reduce para listas.
- Conceptos listos para React.