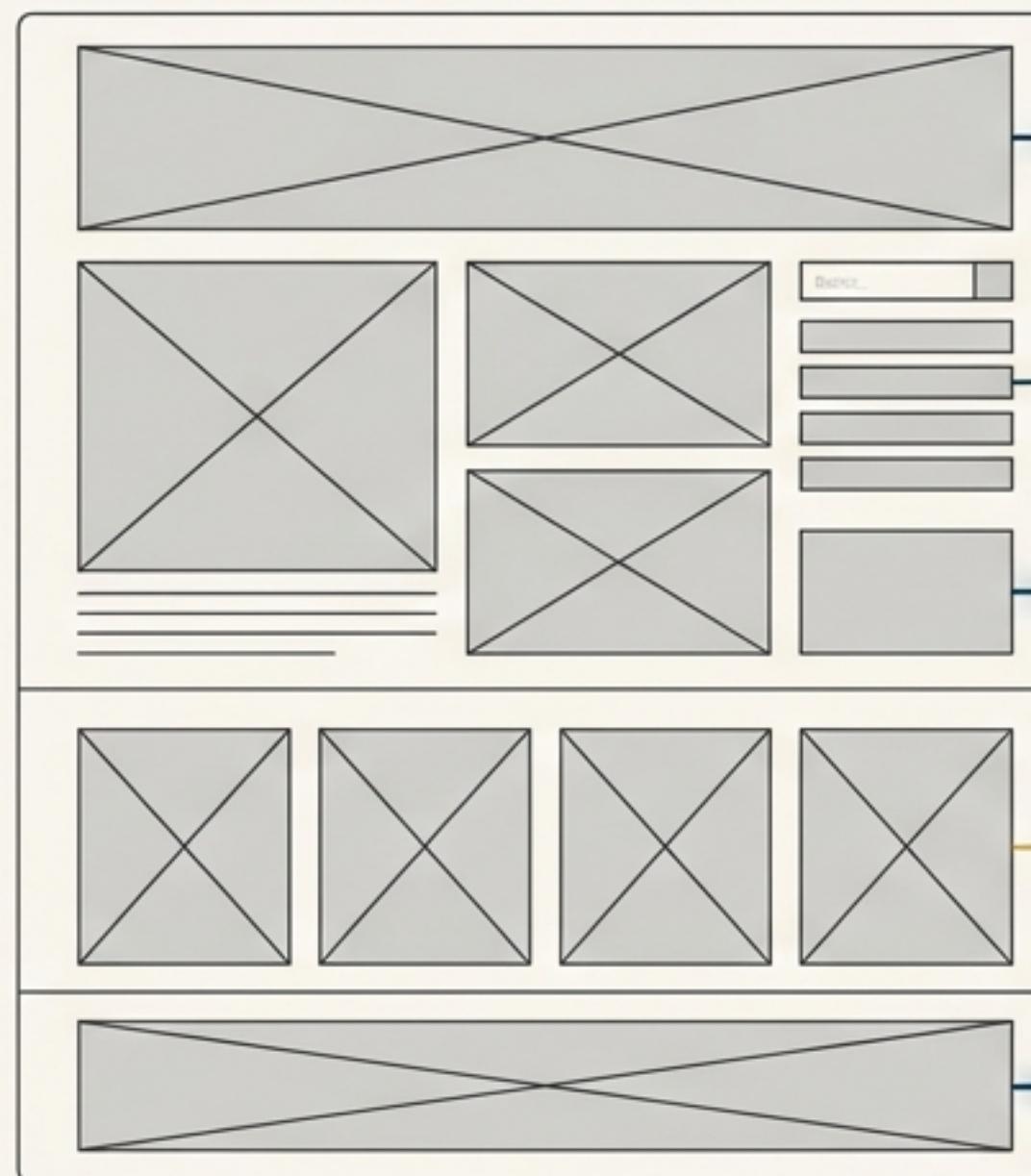
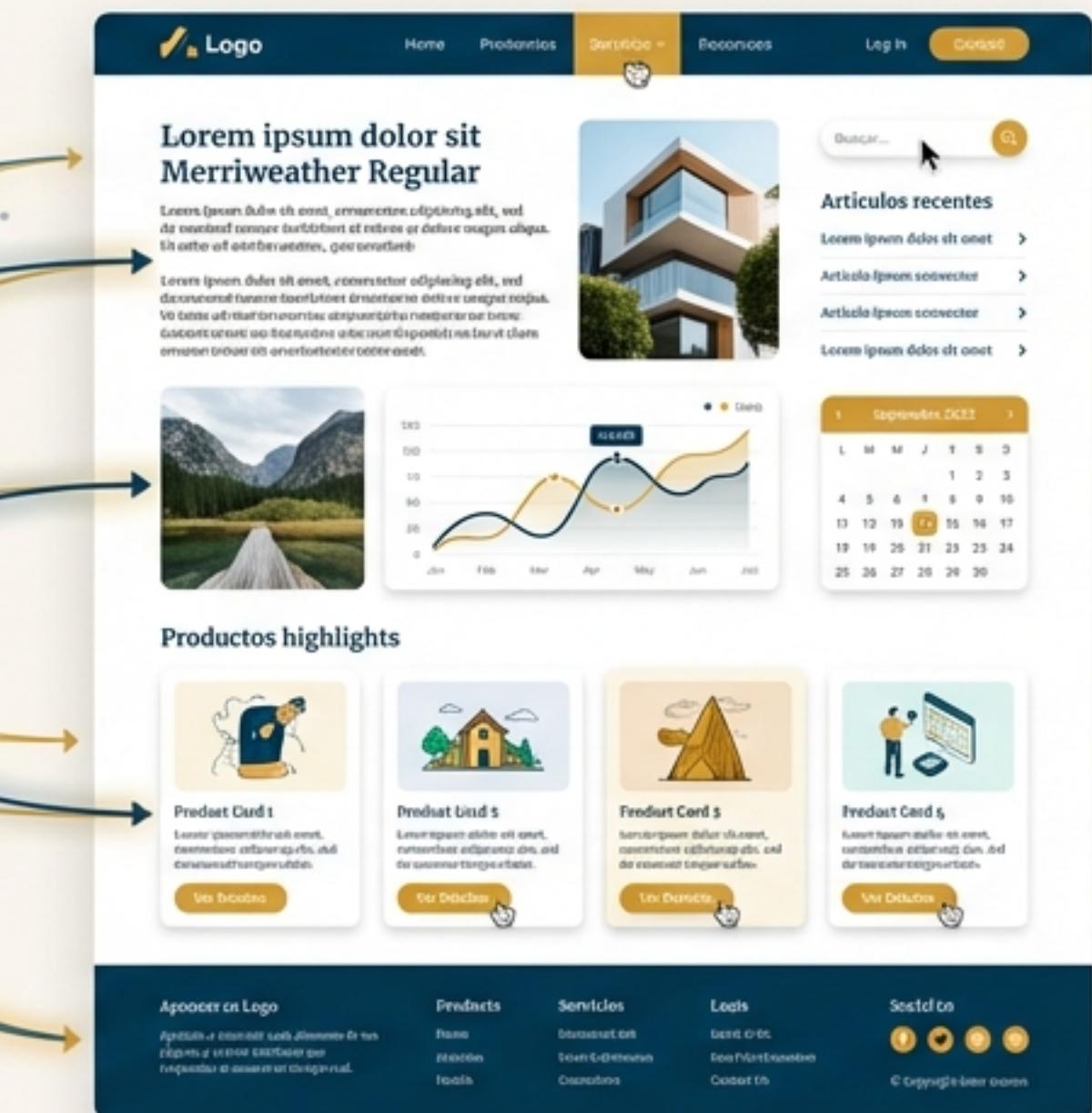


# Módulo 9: Manipulación del DOM

## El Arte de Dar Vida a la Web



Estructura (HTML)

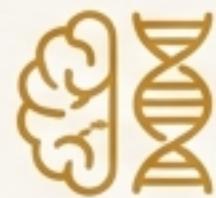


Vida (JavaScript)

Aprende a controlar cada elemento de tus páginas y a crear interfaces que respondan al usuario en tiempo real.

# Tu Camino Hacia la Maestría del DOM

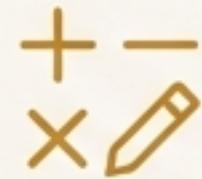
Al finalizar este módulo, serás capaz de construir interfaces de usuario complejas y dinámicas. Dominarás las técnicas esenciales para manipular la web a tu antojo.



Entender el modelo DOM y su estructura como el ADN de una página.



Seleccionar cualquier elemento con la precisión de un cirujano.



Crear, modificar y eliminar elementos sobre la marcha.



Manipular atributos, estilos y clases para transformar la apariencia.



Implementar *event delegation* para un código más limpio y eficiente.



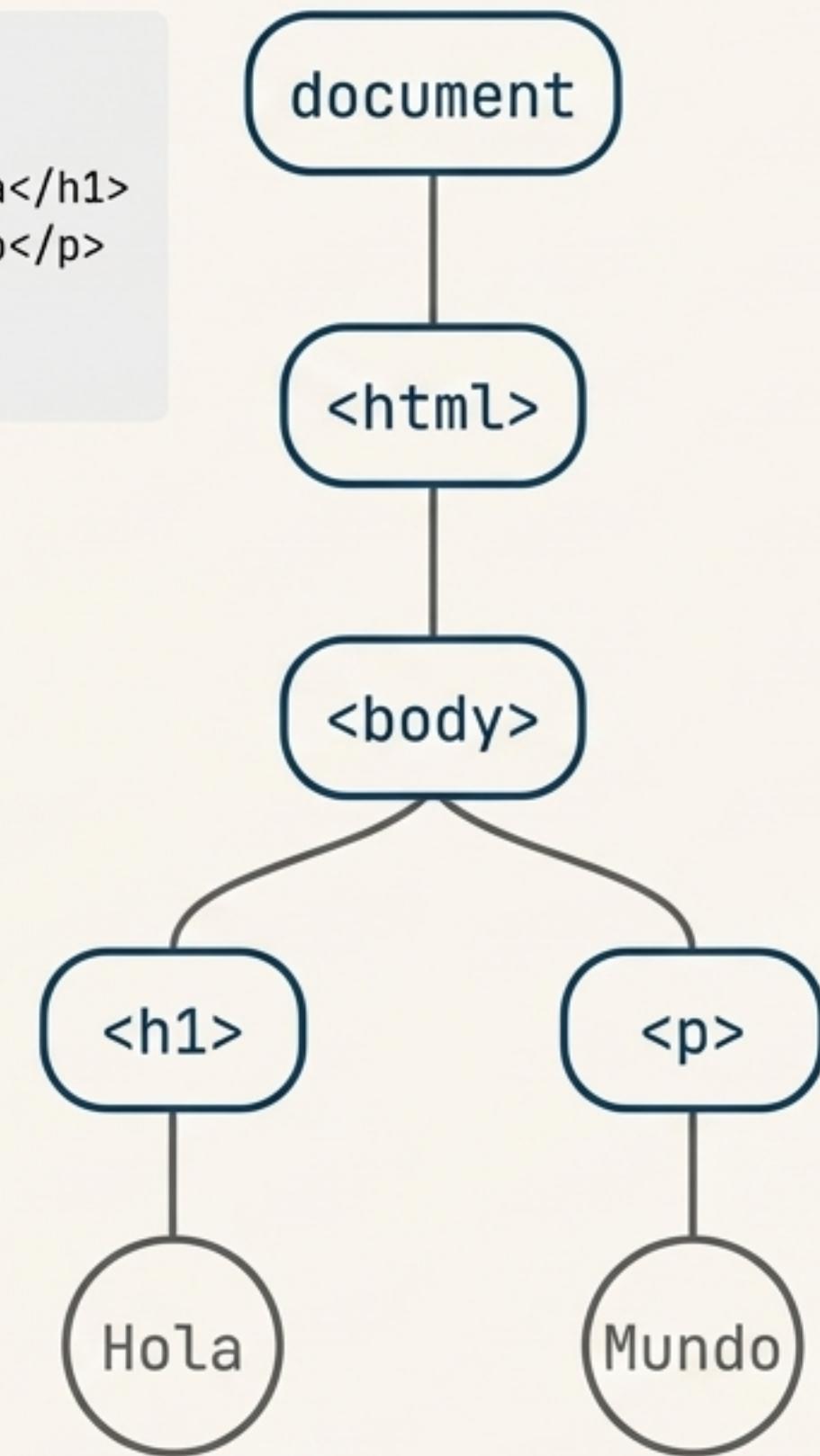
**Competencia Alcanzada:** Utiliza el modelo DOM para la interfaz de programación de aplicaciones en base a estándares establecidos.

# ¿Qué es el DOM? El Plano de la Página Web

El **Document Object Model (DOM)** es la representación de tu documento HTML en la memoria. Es una API que nos permite leer y manipular el contenido, la estructura y los estilos de una página usando JavaScript.

- **No es JavaScript:** Es un estándar del W3C. JavaScript es el lenguaje que usamos para interactuar con él.
- **Estructura de Árbol:** Todo en la página es un *nodo* (elementos, atributos, texto).
- **El Objeto document:** Es el punto de entrada a todo el DOM, la raíz de nuestro árbol.

```
<html>
  <body>
    <h1>Hola</h1>
    <p>Mundo</p>
  </body>
</html>
```



# Herramientas de Precisión: La Selección de Elementos

Para manipular un elemento, primero debes encontrarlo. El DOM nos ofrece un arsenal de métodos, pero dos de ellos son los más potentes y flexibles.

## querySelector()

Devuelve el **primer** elemento que coincide con un selector CSS. Es rápido, específico y versátil.

```
const header = document.querySelector('#main-header');
```

## querySelectorAll()

Devuelve **todos** los elementos que coinciden con un selector CSS en una `NodeList` estática. Ideal para operar en múltiples elementos.

```
const buttons = document.querySelectorAll('.btn-primary');
```

Método
<code>querySelectorAll()</code>
Devuelve <b>todos</b> los elementos que coinciden con un selector CSS en una <code>NodeList</code> estática. Ideal para operar en múltiples elementos.

```
const buttons = document.querySelectorAll('.btn-primary');
```

Método	Retorna	Live/Static	Flexibilidad
<code>getElementById()</code>	Elemento	Live	★
<code>getElementsByClassName()</code>	HTMLCollection	Live	★★
<code>getElementsByTagName()</code>	HTMLCollection	Live	★★
<code>querySelector()</code>	Elemento	Static	★★★★★
<code>querySelectorAll()</code>	NodeList	Static	★★★★★

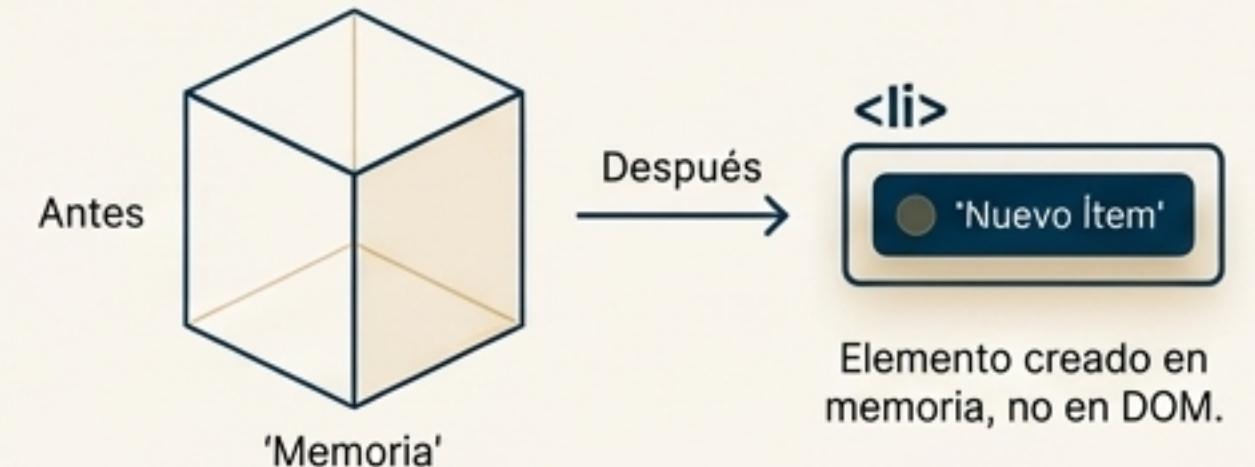
 **Consejo de Maestro:** Prioriza `querySelector` y `querySelectorAll`. Su sintaxis basada en selectores CSS es consistente y te permite seleccionar cualquier cosa que puedas estilizar, dándote un poder inmenso.

# La Caja de Herramientas: Creando y Destruyendo Elementos

## Creación (`createElement`)

**Descripción\*\*:** El primer paso para añadir contenido dinámico.

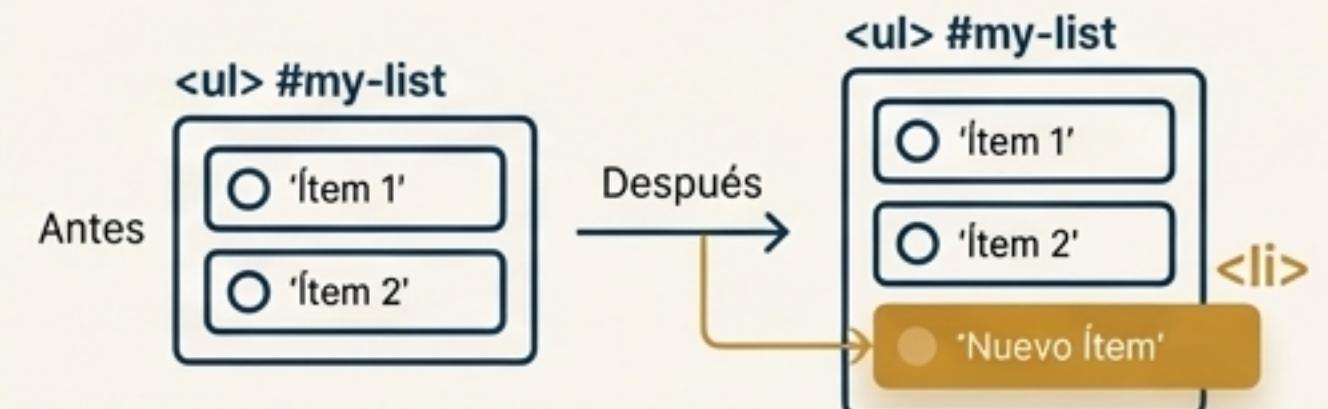
```
// 1. Crear el elemento en memoria  
const newLi = document.createElement('li');  
  
// 2. Añadirle contenido  
newLi.textContent = 'Nuevo ítem';  
  
// 3. Añadirle una clase  
newLi.classList.add('list-item');
```



## Inserción (Métodos para añadir)

**Descripción\*\*:** Una vez creado, debes colocarlo en el DOM.

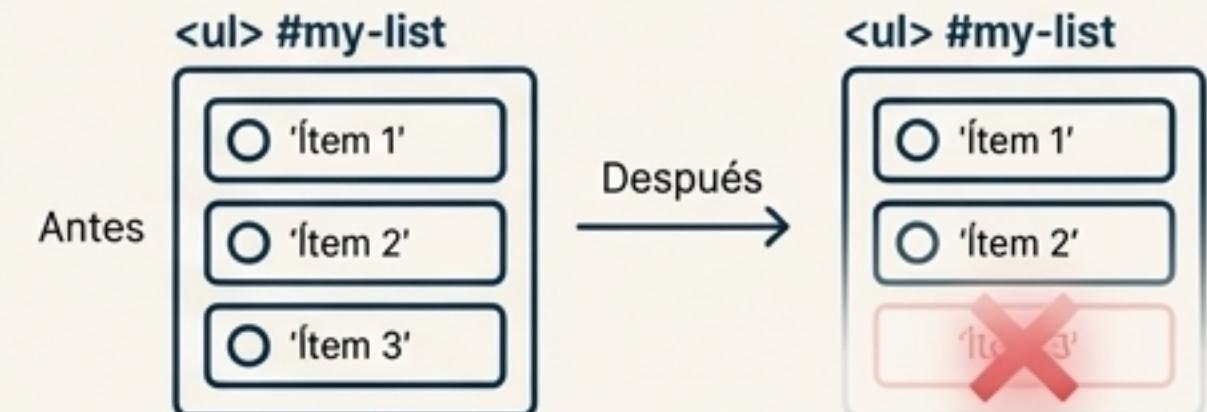
```
// Seleccionar el padre  
const parentUl = document.querySelector('#my-list');  
  
// Añadirlo al final  
parentUl.appendChild(newLi);
```



## Eliminación (`remove`)

**Descripción\*\*:** La forma moderna y sencilla de eliminar un elemento.

```
// Seleccionar el elemento a eliminar  
const itemToRemove = document.querySelector('#item-3');  
  
// ¡Simplemente eliminalo!  
itemToRemove.remove();
```



# Dando Forma: Modificando Contenido, Atributos y Estilos

## Modificar Contenido

**`textContent` (Recomendado):** La forma más segura y rápida de cambiar solo el texto de un nodo. Interpreta todo como texto plano.

**`innerHTML`:** Permite insertar HTML. Es potente pero puede ser lento y abre brechas de seguridad (XSS) si no se usa con cuidado.

### Buena Práctica

Usa `textContent` por defecto.

Recurre a

Recurre a `innerHTML` solo cuando necesites insertar HTML de forma controlada.

## Manipular Atributos

**Uso General:** `getAttribute('href')`, `setAttribute('src', 'new-image.jpg')`.

**Data Attributes:** Una forma excelente de guardar información en el HTML para ser usada por JavaScript.

**HTML:**

```
<div id="user" data-user-id="123" data-role="admin"></div>
```

**JS:**

```
user.dataset.userId; // "123"
```

## Dominar Estilos con `classList` (Recomendado)

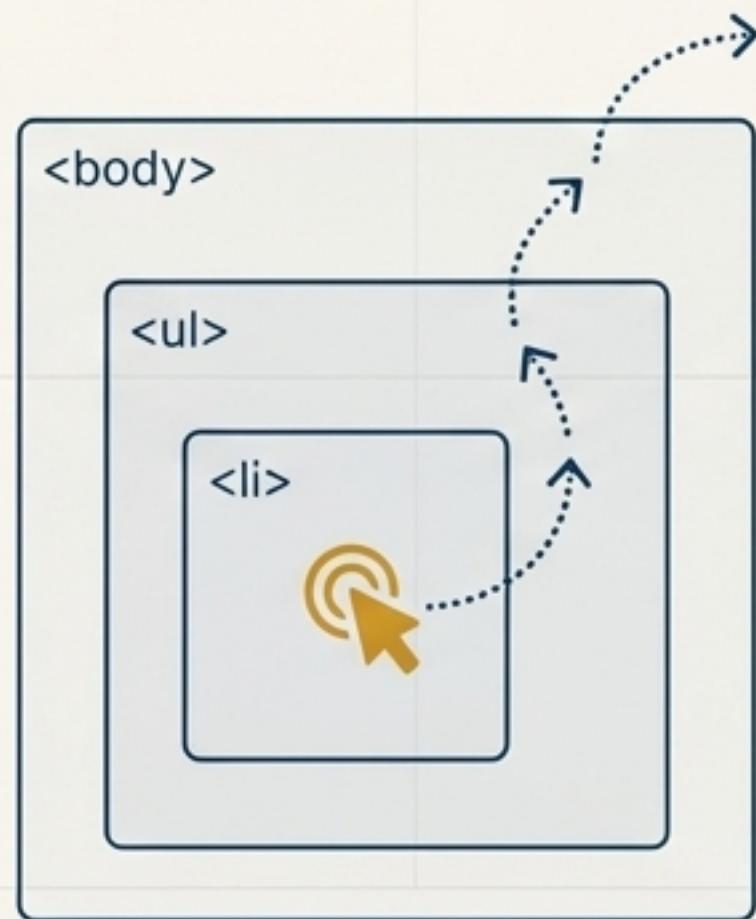
**Descripción:** Olvida la manipulación de estilos inline (`element.style`). `classList` es la forma profesional de gestionar clases CSS.

- + element.classList.add('active')
- element.classList.remove('hidden')
- toggle element.classList.toggle('highlight')
- contains element.classList.contains('selected')

# El Flujo de la Interacción: Event Bubbling y Delegation

## Sección 1: Event Bubbling (Propagación)

Cuando un evento ocurre en un elemento (ej. un `click` en un `- `), se "propaga" hacia arriba a través de sus ancestros (`
`, ``, ``).



## Sección 2: Event Delegation (La Técnica Profesional) ★

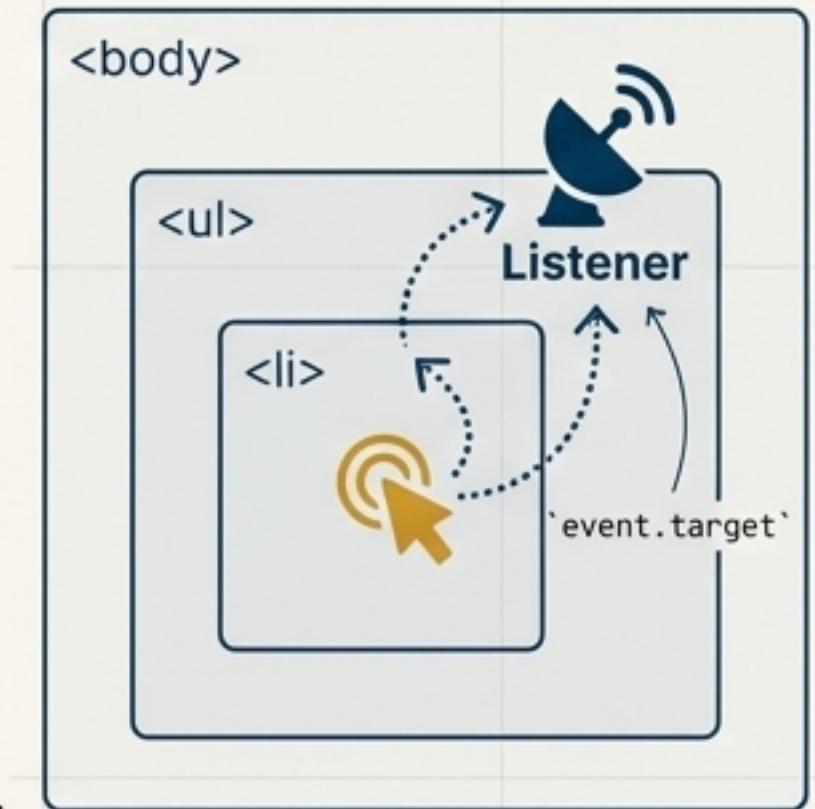
### The Problem

¿Cómo manejamos eventos para 100 ítems de una lista? ¿O para ítems que se añaden dinámicamente? Añadir un \*listener\* a cada uno es ineficiente.

### The Solution

En lugar de eso, colocamos **un solo event listener** en el contenedor padre (ej. el 

).



### Beneficios Clave

- Mejor rendimiento, código más limpio y funciona automáticamente para elementos añadidos en el futuro.

# El Taller del Maestro: Poniendo la Teoría en Práctica

Es hora de construir. Aplicaremos todo lo aprendido para crear componentes interactivos. Para cada ejercicio, seguiremos una “Hoja de Ruta” que desglosa el problema en pasos lógicos.

---

## Desafío 1: Manipulador de Lista (CRUD Básico)

### Hoja de Ruta

Objetivo: Añadir, eliminar y modificar ítems de una lista.

1. **Estructura HTML:** Un `<input>`, un botón de "Añadir" y un `<ul>`.
2. **Selección JS:** Captura los 3 elementos clave.
3. **Lógica de Añadir:** Crea una función que lee el `input`, usa `createElement('li')`, asigna el `textContent` y lo añade al `<ul>` con `appendChild`.
4. **Lógica de Eliminar:** Usa *Event Delegation* en el `<ul>`. Si el `event.target` es un botón "Eliminar", usa `event.target.parentElement.remove()` para borrar el `<li>` padre.

The screenshot shows a user interface for a list manipulation application. At the top right is a blue button labeled "Añadir". To its left is a text input field containing "Nuevo ítem...". Below this is a list of three items, each enclosed in a rounded rectangle. The first item is "Aprender JavaScript" with a trash icon to its right. The second item is "Construir Componentes" with a yellow "Eliminar" button to its right. The third item is "Dominar Eventos" with another yellow "Eliminar" button to its right.

# Taller Práctico: Aplicación de Tareas (Estado y Persistencia)

The screenshot shows a user interface for a task application. At the top, there is a text input field labeled "Añadir nueva tarea..." and a blue "Añadir" button. Below this, there is a list of tasks:

- ✓ Estudiar el DOM
- Implementar Event Delegation
- Practicar con localStorage

Each task has a small trash can icon to its right. At the bottom of the list are three filter buttons: "Todas" (highlighted in dark blue), "Activas", and "Completadas".

## Desafío 2: Lista de Tareas Interactiva

**Objetivo:** Crear una app de tareas con añadir, completar, eliminar, filtrar y guardar los datos.

**Hoja de Ruta para la Solución:**

- Estructura HTML:** Input para nuevas tareas, contenedor `<ul>` para la lista y botones de filtro (Todas, Activas, Completadas).
- Selección de Elementos:** Captura el `input`, el `<ul>` y los botones de filtro.
- Lógica de Añadir Tarea:** Crea un `<li>` con un `<span>` para el texto y un botón de 'Eliminar'. Usa `createElement` y `appendChild`. Añade la nueva tarea a un array que representa el estado. Llama a la función de guardado.
- Lógica de Completar/Eliminar:**
  - Usa **Event Delegation** en el `<ul>`. Si se hace clic en el `<li>`, usa `classList.toggle('completed')`. Si se hace clic en el botón 'Eliminar', usa `.remove()` en el `<li>`. Actualiza el array de estado y guarda.
- Persistencia con `localStorage`:**
  - `guardarTareas()`: Convierte el array de tareas a JSON (`JSON.stringify`) y guárdalo en `localStorage`.
  - `cargarTareas()`: Al iniciar, lee el JSON de `localStorage`, lo convierte de nuevo a un array (`JSON.parse`) y renderiza la lista.

# Taller Práctico: Construyendo Componentes de UI Esenciales

## Desafío 3: Tabs Dinámicos



### Objetivo

Crear pestañas que cambian el contenido visible sin recargar la página.

### Hoja de Ruta

#### 1. Estructura

Un contenedor para los botones de las pestañas y otro para los paneles de contenido. Usa `data-tab` attributes para conectar cada botón con su panel.

#### 2. Lógica de Eventos

Añade un *listener* al contenedor de los botones (Event Delegation).

#### 3. Función de Cambio

Al hacer clic, quita la clase `'active'` de todas las pestañas y paneles. Añade la clase `'active'` solo a la pestaña clickeada y a su panel de contenido correspondiente (identificado por el `'data-tab'`).

## Desafío 4: Modal (Ventana Emergente)



### Objetivo

Mostrar y ocultar un modal, manejando el foco y el fondo.

### Hoja de Ruta

#### 1. Estructura

Un botón 'Abrir Modal', un `'<div>' para el overlay de fondo y el '<div>' del modal (con un botón de cierre 'X'). Inicialmente ocultos con CSS.`

#### 2. Lógica de Apertura

El botón 'Abrir Modal' añade una clase `'is-visible'` al overlay y al modal. Opcional: añade `'overflow: hidden'` al `'<body>'`.

#### 3. Lógica de Cierre

El botón 'X' y el overlay tienen *listeners* que quitan la clase `'is-visible'`. Añade un *listener* al `'document'` para el evento `'keydown'`. Si `'event.key === 'Escape''`, cierra el modal.

# Taller Práctico: Aplicaciones Avanzadas y Dinámicas

## Desafío 5: Galería de Imágenes Interactiva



### Objetivo

Cargar, filtrar y mostrar imágenes en un lightbox.

### Hoja de Ruta

- Renderizado Dinámico:** Usa un array de objetos de imágenes. Recórrelo con `.map()` para generar el HTML de cada imagen (`createElement` o *template literals*) y añádelo al DOM. Usa `data-category` en cada imagen.
- Filtros:** Los botones de filtro (ej. "Naturaleza", "Ciudad") tienen un `data-filter`. Al hacer clic, filtra el array de imágenes y vuelve a renderizar la galería solo con los resultados.
- Lightbox:** Usa *Event Delegation* en el contenedor de la galería. Al hacer clic en una imagen, abre un modal (Lightbox) y muestra la imagen en grande.

## Desafío 6: Carrito de Compras Dinámico

	Auriculares Inalámbricos \$120.00	<input type="button" value="-"/> <input checked="" type="button" value="1"/> <input type="button" value="+"/> <input type="button" value="Delete"/>
	Producto Nombre \$120.00	<input type="button" value="-"/> <input checked="" type="button" value="1"/> <input type="button" value="+"/> <input type="button" value="Delete"/>
	Carné de Tarjeta \$80.00	<input type="button" value="-"/> <input checked="" type="button" value="1"/> <input type="button" value="+"/> <input type="button" value="Delete"/>
Subtotal: \$120.00		
Impuestos: \$12.00		
<b>Total: \$132.00</b>		

### Objetivo

Añadir productos, actualizar cantidades y calcular el total automáticamente.

### Hoja de Ruta

- Gestión de Estado:** Mantén el carrito como un array de objetos en JavaScript (ej. `[{ id, nombre, precio, cantidad }]`  ).
- Renderizado:** Crea una función `renderCart()` que borre el contenido actual del carrito en el DOM y lo reconstruya a partir del array de estado. Esto asegura que la UI siempre refleje los datos.
- Interacción:** Usa *Event Delegation* en la lista de productos y en el carrito para manejar los botones de "Añadir", "Eliminar" o "Cambiar Cantidad".
- Cálculo de Total:** Despues de cualquier cambio, vuelve a calcular el total usando `Array.reduce()` sobre el array del carrito y actualiza el elemento del total en el DOM con `textContent` .

# Anatomía de una Aplicación Reactiva: El Carrito de Compras

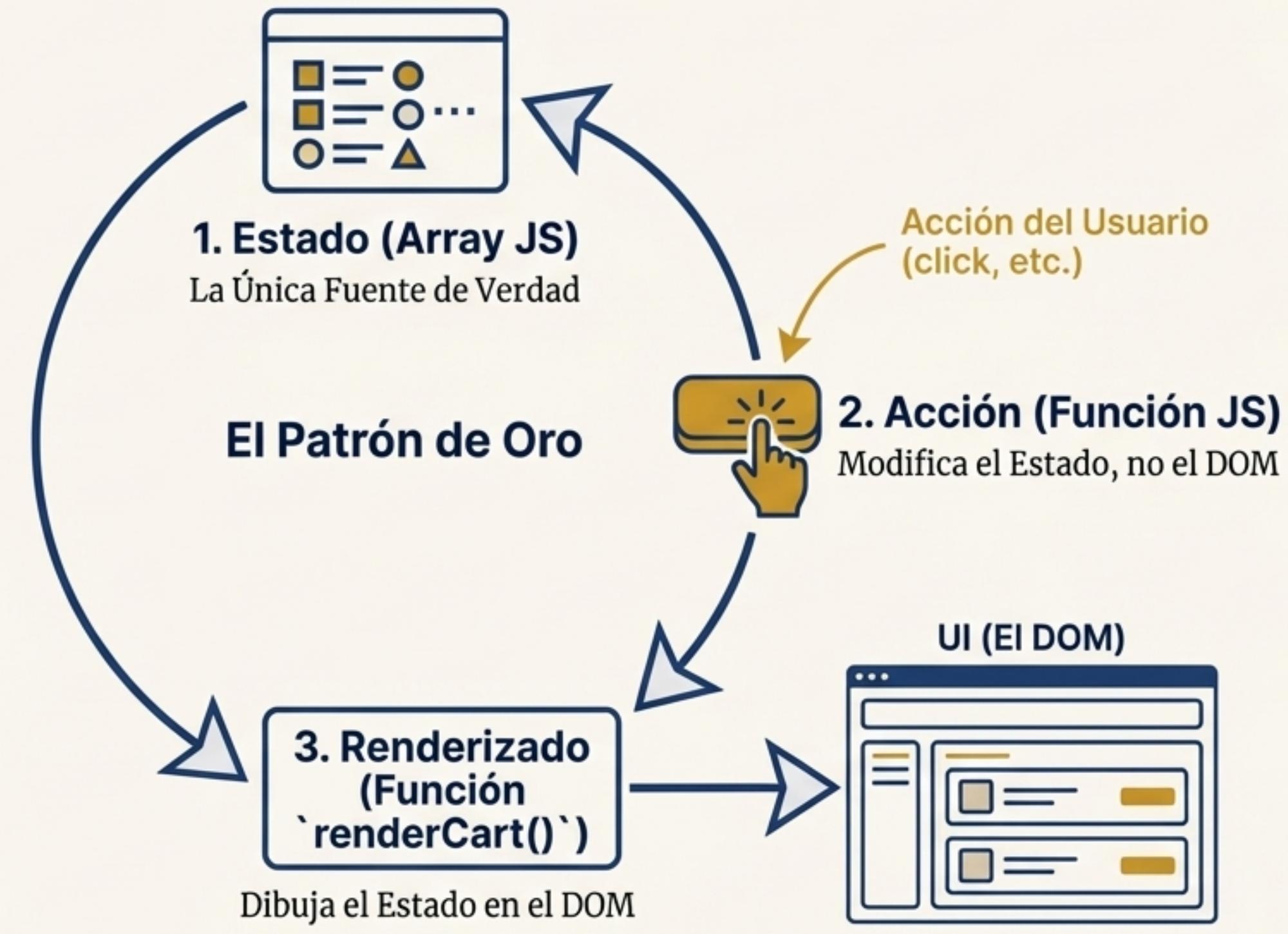
Un carrito de compras es un micro-sistema 'reactivo'. La clave es separar los datos (el estado) de la visualización (el DOM). La UI es solo un reflejo del estado.

## Código Esquemático

```
let cart = [] // 1. El Estado

function addToCart(item) {
  cart.push(item) // 2. La Acción modifica el estado
  renderCart() // 3. Se vuelve a renderizar
  updateTotal()
}

function renderCart() {
  // Limpia el DOM
  // Recorre el array 'cart' y crea los elementos HTML
}
```



**Conclusión:** Este patrón (Estado → Acción → Renderizado) es la base de frameworks modernos como React y Vue. Dominarlo te prepara para el siguiente nivel.

# Tu Certificación de Maestro del DOM

¡Felicitaciones! Has completado el camino. No solo has aprendido la teoría, sino que la has puesto en práctica para construir aplicaciones web interactivas y robustas.

## Checklist de Habilidades Dominadas

- Entiendes la estructura del DOM como un árbol de nodos.
- Seleccionas cualquier elemento con la precisión de `queryS`.
- Navegas por el árbol del DOM con soltura.
- Modificas contenido de forma segura con `textContent`.
- Creas, eliminas y reemplazas elementos dinámicamente.
- Manipulas atributos y `data attributes` para enriquecer tu HTML.
- Usas `classList` como un profesional para gestionar estilos.
- Implementas *event delegation* para un código eficiente y escalable.
- Has completado los 6 desafíos prácticos, desde un simple CRUD hasta un carrito de compras.

## Mirando Hacia Adelante

Has sentado las bases. Ahora estás listo para el siguiente nivel de interacción con el usuario.

Siguiente:: Módulo 10: Formularios HTML5 + JavaScript →