**Parte Teórica**[¶](https://kevin-117.github.io/sis-414-2025-web/practicas/Practica%202%20-%20DOM/#parte-teorica)

**1. Preguntas Conceptuales**

**a)** ¿Qué es el DOM y cómo se relaciona con HTML?

El DOM es el **puente** entre el **HTML estático** y el **JavaScript dinámico**, permitiendo que las páginas web sean interactivas y respondan a las acciones del usuario

**b) Explica la diferencia entre:**

* **document.getElementById() vs document.querySelector().**
* **textContent vs innerHTML.**
* **Selectores**:
  + getElementById es ideal para IDs.
  + querySelector es versátil pero menos eficiente en casos simples.
* **Manipulación de contenido**:
  + textContent es seguro y rápido para texto.
  + innerHTML es poderoso pero requiere precaución.

Ambas herramientas son esenciales en el desarrollo web, pero su uso depende del contexto y necesidades de seguridad

**c) ¿Para qué sirve addEventListener? Proporciona un ejemplo.**

addEventListener es un método en JavaScript que permite **asociar una función (oyente/event handler) a un evento específico** de un elemento del DOM. Su propósito es manejar interacciones del usuario (clics, teclas, movimientos del mouse, etc.) o eventos del navegador (carga de página, resize, etc.).

Ej:

const boton = document.getElementById("miBoton");

const mensaje = document.getElementById("mensaje");

// Añadir un event listener para el evento 'click'

boton.addEventListener("click", function() {

mensaje.textContent = "¡Botón clickeado!";

});

**d) ¿Qué métodos del DOM se usan para capturar valores de un formulario?**

Para obtener valores de un formulario en JavaScript, puedes usar los siguientes métodos y propiedades del DOM:

getElementById()**+**.value  
Ideal para inputs con id.

querySelector()**+**.value  
Útil para cualquier selector CSS (clases, atributos, etc.).

**e) Explica cómo prevenir el envío por defecto de un formulario con JavaScript.**

Para prevenir el envío de un formulario:

* Agrega un listener al evento submit.
* Usa e.preventDefault() para cancelar la acción por defecto.
* Procesa los datos manualmente (validación, AJAX, etc.).

**f) ¿Qué es el "almacenamiento en memoria" y en qué se diferencia de localStorage?**

Guarda datos **temporalmente** en la RAM del navegador mientras la pestaña o aplicación está abierta.

* **Memoria**: Velocidad y volatilidad. Ideal para datos efímeros.
* localStorage: Persistencia y accesibilidad. Perfecto para datos a largo plazo.

**2. ANALISIS DE CODIGO.**

1. **¿Qué hace el código?**

Selecciona el botón con id="btn" y le añade un **event listener** para el evento click.

Cuando se hace clic en el botón, el texto del elemento <p id="mensaje"> se actualiza a "¡Botón presionado!" usando textContent.

Al hacer clic en el botón, aparece el mensaje en el párrafo.

1. **¿Qué pasaría si cambiamos textContent por innerHTML?**

* Usa textContent cuando:

Solo necesitas mostrar texto plano.

Priorizas seguridad (evitar XSS).

* Usa innerHTML cuando:

Requieres renderizar HTML dinámico (ej: negritas, enlaces).

**Solo si confías en el origen del contenido** (nunca con input de usuario sin sanitizar).

**3. ANALISIS DE CODIGO.**

**a) ¿Qué hace el código al enviar el formulario?**

Cada vez que se envía el formulario, el nombre ingresado aparece en la lista.

**b). ¿Cómo se simula la "persistencia de datos" aquí?**

Captura nombres de un formulario y los muestra en una lista.

Usa un array en memoria como "almacenamiento temporal".

preventDefault(): Clave para aplicaciones dinámicas sin recargas.

**Persistencia simulada**: Solo válida durante la sesión actual.