DOM

DOM significa Document Object Model (Modelo de Objetos del Documento)

Es la representación en memoria (en forma de árbol de objetos) que el navegador crea cuando carga una página HTML.

Cada etiqueta HTML se convierte en un objeto JS manipulable:

- Un se convierte en un objeto HTMLDivElement.
- Un <p> se convierte en HTMLParagraphElement.
- Y así sucesivamente.

Entonces, **cuando tu HTML se carga**, el navegador lo convierte internamente en algo parecido a esto:

```
<html>
 <head>...</head>
 <body>
  <h1>Hola</h1>
  <p id="mensaje">Bienvenido</p>
  <button>Saludar</button>
 </body>
</html>
```

El navegador lo traduce internamente a un **árbol DOM**:

```
document
└── html
    ├── head
    └── body
        ├── h1
        ├── p#mensaje
        └── button
```

Y ese árbol es lo que tú puedes **recorrer y modificar desde JavaScript**.

Resumiendo, el DOM es la manera que tiene JavaScript de ver y manipular el HTML que ya está cargado en la página.

¿Para qué se usa el DOM en la práctica?

En el ejemplo llamado “uso_basico_DOM” del aula virtual, vamos a ir viendo las formas más comunes de utilizar el DOM.

Eventos:

Un evento NO es simplemente un “clic”, un “submit” o un “keydown”. Un evento es un objeto generado por el navegador cuando “algo ocurre” en la página.

Ejemplos de acciones que pueden generar un evento:

- el usuario hace clic,
- pulsa una tecla,
- mueve el ratón,
- un formulario intenta enviarse,
- una imagen termina de cargar,
- etc.

Cada vez que uno de estos sucesos ocurre, el navegador crea un **objeto evento** con toda la información sobre lo sucedido y lo entrega a la función que tú hayas definido.

Ejemplo de un objeto evento real:

```
{
  "type": "click",
  "target": "<button id='borrar'>",
  "clientX": 133,
  "clientY": 420,
  "timeStamp": 12345
}
```

Ese objeto contiene toda la información útil para trabajar.

Dentro de las funciones que solemos utilizar, destaca addEventListener, una función que se ejecuta cuando ocurre un evento dentro de la página: un clic, un envío de formulario, pulsar una tecla, mover el ratón, cargar la página, etc.

Ojo, no es la única forma de ejecutar código cuando ocurre un evento

MÉTODO ANTIGUO → onclick= (menos recomendable)

En HTML:

```
<button onclick="saluda()">Hola</button>
```

En JS:

```
function saluda() {
  alert("Hola");
}
```

MÉTODO MODERNO → addEventListener (el estándar actual)

En HTML:

<button Id="btn">Hola</button>

En JS:

document.getElementById("btn").addEventListener("click", saluda);

DIFERENCIAS:

onclick (menos recomendado)	**addEventListener (recomendado)**

Solo permite registrar **un evento**
btn.onclick = fn1;
btn.onclick = fn2;
El segundo **machaca** al primero.
	Puedes registrar tantos como quieras :
	btn.addEventListener("click", fn1);
	btn.addEventListener("click", fn2);

Mezcla HTML y JS	Separa estructura y lógica
Más difícil de mantener	Estándar moderno
No funciona bien con módulos, frameworks o AJAX	Total compatibilidad

A partir de ahora nos centraremos únicamente en addEventListener, que será la función que utilicemos.

Como hemos dicho, cada vez que el usuario hace algo en la página —clic, movimiento de ratón, tecla, scroll, enviar formulario— el navegador genera un objeto muy completo que describe exactamente lo que ocurrió.

Ese objeto es lo que llamamos evento, y se entrega automáticamente a tu función:

```
element.addEventListener("click", (e) => {
   // aquí tienes toda la información del evento
});
```

e se podría haber llamado de cualquier otra manera, **y NO es obligatorio**, pero si lo usas tienes acceso a datos imprescindibles: qué elemento se pulsó, coordenadas, teclas, si el usuario tenía pulsado Ctrl, etc.

Ejemplos de usos útiles de e:

e.type — Qué evento ocurrió

Muy útil si quieres una sola función para varios eventos.

Ejemplo: un botón que cambia al pasar el ratón o hacer clic

```
boton.addEventListener("mouseenter", (e) => manejarBoton(e));
boton.addEventListener("click", (e) => manejarBoton(e));
```

```
function manejarBoton(e) {
  if (e.type === "mouseenter") {
    boton.textContent = "¡Hola!";
  }
  if (e.type === "click") {
    boton.textContent = "Click hecho";
  }
}
```

e.target — El elemento donde ocurrió el evento

Esto es importantísimo cuando **clicas una cosa**, pero **el listener está en otra distinta**.

Esto puede ser lógico: Si tienes una lista con 10 elementos y haces clic en uno…no quieres poner 10 listeners, uno por cada elemento.

Ejemplo: html

```
<ul id="lista">
  <li>Juan</li>
  <li>Ana</li>
  <li>Carlos</li>
</ul>
```

JavaScript:

```
lista.addEventListener("click", (e) => {
  e.target.style.color = "red";
});
```

Lo que pasa es que clicas en “Ana”, y e.target es exactamente ese <li>, por lo que es el que se pone en rojo

Aunque el *listener* está en <ul>, el evento ocurrió en el <li>.

e.preventDefault() — Evitar la acción automática del navegador

Hay elementos que tienen un comportamiento por defecto, como por ejemplo:

- los formularios recargan la página
- los enlaces navegan a otra página

Si por ejemplo no queremos que un formulario recargue la página, ejecutamos:

```
formulario.addEventListener("submit", (e) => {
        e.preventDefault(); // evita que se envíe al servidor
        alert("Formulario tratado sin recargar página!");
    }
});
```