# PRACTICA 1: MANEJO ESTRUCTURA DOM

## INTRODUCCIÓN

### 1. QUÉ ES EL DOM

El **DOM** convierte el documento HTML en una estructura de objetos con la que puedes interactuar mediante código. Debemos pensar en el DOM como un árbol de objetos donde cada elemento del HTML (como , <p>, <h1>, etc.) es un nodo dentro de ese árbol.

Por ejemplo, con el siguiente código html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Mi página web</title>
</head>
<body>
  <h1>Bienvenido a mi página</h1>
  <p>Este es un párrafo.</p>
  <button>Haz clic aquí</button>
</body>
</html>
```

En el DOM se representaría de la siguiente forma:

```
document (raíz)
└── html
    ├── head
    │   └── title
    └── body
        ├── h1
        ├── p
        └── button
```

JavaScript puede acceder y modificar los elementos del DOM mediante varias funciones y métodos. Aquí te explico algunos de los más comunes:

- Acceder a elementos del DOM Para acceder a un elemento HTML en el DOM, puedes usar varios métodos:

- **document.getElementById(id):** Devuelve el primer elemento con el ID especificado.
- **document.getElementsByClassName(className)**: Devuelve todos los elementos con la clase especificada.
- **document.getElementsByTagName(tagName)**: Devuelve todos los elementos con el nombre de etiqueta especificado.
- **document.querySelector(selector):** Devuelve el primer elemento que coincide con el selector CSS proporcionado.

- **document.querySelectorAll(selector):** Devuelve todos los elementos que coinciden con el selector CSS.

Una vez que se accede a los elementos podemos cambiar el contenido del nodo, modificar atributos, añadir o eliminar un nuevo elemento, etc.

## 2. MANEJAR LA ESTRUCTURA DOM

**Cómo seleccionar otros nodos según algunas de las relaciones de parentesco establecidas alrededor de tal elemento.**

- **parentNode:** Por medio de **parentNode** podemos seleccionar el elemento padre de otro elemento.
- f**irstChild:** Con **firstChild** lo que seleccionamos es el primer hijo de un elemento. Por desgracia, hay discrepancias entre los diversos navegadores sobre qué debe considerarse o no hijo de un nodo, por lo que esta propiedad en ocasiones complica demasiado un script.
- **lastChild:** La propiedad **lastChild** funciona exactamente como **firstChild**, pero se refiere el último de los hijos de un elemento. Se aplican, por tanto, las mismas indicaciones anteriores.
- **nextSibling:** Gracias a nextSibling, lo que podemos seleccionar es el siguiente hermano de un elemento. Se aplican las mismas limitaciones que para las dos propiedades anteriores.
- **previousSibling : previousSibling** funciona igual que nextSibling, pero selecciona el hermano anterior de un elemento

## 3. CREACIÓN DE ELEMENTOS

- **appendChild:** Por medio de appendChild podemos incluir en un nodo un nuevo hijo, de esta manera:

  elemento_padre.appendChild(nuevo_nodo);

  El nuevo nodo se incluye inmediatamente después de los hijos ya existentes —si hay alguno— y el nodo padre cuenta con una nueva rama.

**Por ejemplo, el siguiente código:** Crea un elemento ul y un elemento li, y convierte el segundo en hijo del primero.

```
const lista = document.createElement('ul');
const item = document.createElement('li');
lista.appendChild(item);
```

- **insertBefore:** Nos permite elegir un nodo del documento e incluir otro antes que él.

  ***elemento_padre.insertBefore(nuevo_nodo,nodo_de_referencia);***

**Por ejemplo, el siguiente código**

```
<div id="padre">
        <p>Un párrafo.</p>
        <p>Otro párrafo.</p>
```

```
</div>
```

y quisiéramos añadir un nuevo párrafo antes del segundo, lo haríamos así:

```
// Creamos el nuevo párrafo
const nuevo_parrafo
=document.createElement('p').appendChild(document.createTextNode('Nuevo
párrafo.'));
// Recojemos en una variable el segundo párrafo
const segundo_p = document.getElementById('padre').getElementsByTagName('p')[1];
// Y ahora lo insertamos
document.getElementById('padre').insertBefore(nuevo_parrafo,segundo_p);
```

- **insertAfter:** No hay un metodo que inserte un nodo detrás de otro
- **replaceChild:** Para reemplazar un nodo por otro contamos con replaceChild, cuya sintaxis es:

  ```
  elemento_padre.replaceChild(nuevo_nodo,nodo_a_reemplazar);
  ```
- **removeChild:** Dado que podemos incluir nuevos hijos en un nodo, tiene sentido que podamos eliminarlos. Para ello existe el método removeChild.

  ```
  elemento_padre.removeChild(nodo_a_eliminar);
  ```
- **cloneNode:** Por último, podemos crear un clon de un nodo por medio de cloneNode:

  ```
  elemento_a_clonar.cloneNode(booleano);
  ```

  El booleano que se pasa como parámetro define si se quiere clonar el elemento —con el valor false—, o bien si se quiere clonar con su contenido —con el valor true—, es decir, el elemento y todos sus descendientes.

## 4. MODIFICAR Y AÑADIR ELEMENTOS EN LA ESTRUCTURA DOM

### ◦ innerHTML

La propiedad Element.innerHTML devuelve o establece la sintaxis HTML describiendo los descendientes del elemento.

Al establecerse se reemplaza la sintaxis HTML del elemento por la nueva.

Para insertar el código HTML en el documento en lugar de cambiar el contenido de un elemento, use el método insertAdjacentHTML().

### ◦ insertAdjacentHTNML

EL método insertAdjacentHTML() de la interfaz Element analiza la cadena de texto introducida como cadena HTML o XML e inserta al árbol DOM los nodos resultantes de dicho análisis en la posición especificada. Este método no re-analiza el elemento sobre el cual se está invocando y por lo tanto no corrompe los elementos ya existentes dentro de dicho elemento. Esto evita el paso adicional de la serialización, haciéndolo mucho más rápido que la manipulación directa con innerHTML

Sintaxis

***element.insertAdjacentHTML(posición, texto);***

Parámetros

- **Posición** Un DOMString que representa la posición relativa al elemento, y deberá ser una de las siguientes cadenas:
  - 'beforebegin': Antes que el propio elemento.
  - 'afterbegin': Justo dentro del elemento, antes de su primer elemento hijo.
  - 'beforeend': Justo dentro del elemento, después de su último elemento hijo.
  - 'afterend': Después del propio elemento.
- **Texto** Es la cadena a ser analizada como HTML o XML e insertada en el árbol.

**Nota: Las posiciones beforebegin y afterend funcionan únicamente si el nodo se encuentra en el árbol DOM y tiene un elemento padre.**

## 5. INTRODUCCIÓN A LOS EVENTOS

En JavaScript, la interacción con el usuario se consigue mediante la captura de los eventos que éste produce. Un **evento** es una acción del usuario ante la cual puede realizarse algún proceso (por ejemplo, el cambio del valor de un formulario, o la pulsación de un enlace).

Los eventos se capturan mediante los manejadores de eventos. El proceso a realizar se programa mediante funciones JavaScript llamadas por los manejadores de eventos.

Existen varias formas diferentes de manejar eventos en Javascript.

| Forma | Ejemplo |
|---|---|
| Mediante atributos HTML | <button onClick="..."></button> |
| Mediante propiedades Javascript | .onclick = function() { ... } |
| Mediante addEventListener() | addEventListener("click", ...) |

Cada una de estas opciones se puede utilizar para gestionar eventos en Javascript de forma equivalente, pero cada una de ellas tiene sus ventajas y sus desventajas. En los siguientes apartados veremos detalladamente sus características, pero por norma general, lo aconsejable es utilizar la última, los listeners, ya que son las más potentes y flexibles.

1. **Mediante atributos HTML:** Es la opción menos aconsejable ya que suele ser buena práctica **no** incluir llamadas a funciones Javascript en nuestro código .html. No se debe mezclar código html con javascript.

**Ejemplo:**

```
<script>
 function saludar() {
  alert("Hola que tal estas!");
 }
</script>

<button onClick="saludar()">Saludar</button>
```

2. **Mediante propiedades Javascript:** En esta ocasión haremos uso de una propiedad Javascript, a la que le asignaremos la función con el código asociado.

```
<button>Saludar</button>
<script>
const boton = document.querySelector("button");
//Este método devuelve el primer botón. A partir de ese momento se almacena en la constante boton y nos permite acceder a sus métodos

button.onclick = function() {
  alert("Hola que tal estas!");
}
</script>
```

3. **Mediante addEventListener():**
   - Con **.addEventListener()** se pueden añadir fácilmente varias funcionalidades.
   - Con **.removeEventListener()** se puede eliminar una funcionalidad previamente añadida.
   - Con **.addEventListener()** se pueden indicar ciertos comportamientos especiales.

El método addEventListener() permite añadir una escucha del evento indicado (primer parámetro) y en el caso de que ocurra, se ejecutará la función asociada (segundo parámetro)

| Método | Descripción |
|---|---|
| .addEventListener(event,func) | Escucha el evento event, y si ocurre, ejecuta func. |
| .addEventListener(event,func,options) | Idem, pasándole ciertas opciones. |

Para comprobar su funcionamiento, utilizaremos el mismo ejemplo que en los casos anteriores.

```
const boton = document.querySelector("button");
button.addEventListener("click", function() {
  alert("Hola que tal estas!");
});
```

Podemos utilizar funciones para que el código quede mucho más legible.

```
const boton = document.querySelector("button");
const saludar = () => alert("hola que tal estas!");
button.addEventListener("click", saludar);
```

Una de las características más importantes es que nos permite añadir múltiples listeners y así asociar varias funciones a un mismo evento

```
<button>Saludar</button>
<style>
 .red { background: red }
</style>
<script>
        const boton = document.querySelector("button");
        const saludar = () => alert("Hola que tal estas!");
        const cambiar = () => boton.classList.toggle("red"); //Añade o quita el color rojo
        del botón

        boton.addEventListener("click", saludar);       // Hello message
        boton.addEventListener("click", cambiar);       // Add/remove red CSS
</script>
```

Podemos trabajar con varias opciones y eliminar la escucha. Esto se tratará más adelante.

***Ejemplo:***

```
<HTML>
<HEAD><TITLE>Eventos</TITLE>
<SCRIPT>
<!--
function Reacciona(campo) {
        alert("¡Introduzca un valor!")
        campo.focus()
}
//-->
</SCRIPT></HEAD>
<BODY>
<FORM METHOD=POST>
<INPUT TYPE=text NAME=campo onFocus="Reacciona(this)">
</FORM>
 </BODY>
 </HTML>
```

***Nota: El evento onFocus no funciona con Mozilla.***

## Eventos onLoad y onUnload:

El evento onload de Javascript se activa cuando se termina de cargar la página. Se ha de colocar en la etiqueta <body>, aunque en versiones modernas de Javascript, también lo aceptan otros elementos como las imágenes.
Con el evento onload podemos ejecutar acciones justo cuando se han terminado de cargar todos los elementos de la página. Es un evento bastante utilizado pues es muy habitual que se deseen llevar a cabo acciones en ese preciso instante.

**Ejemplo:**

```
<BODY onLoad="Hola()" onUnload="Adios()">
```

La función Hola() se ejecutará antes de que se cargue la página y la función Adios() al abandonarla.

***Ej: Pasar el foco a un elemento del formulario.***

```
<a href="#" onclick="document.getElementById("nombre").focus()">Nombre</a>
<a href="#" onclick="document.getElementById("apellidos").focus()">Apellidos</a>
```

## SECUENCIA Y DESARROLLO:

1. **Ejercicio 1: Prueba el siguiente ejemplo, el cual crea una aplicación web con un cuadro de texto y un botón que, al hacer clic en el botón, añada un nuevo párrafo a la página con el texto que el usuario ha introducido en el campo de texto.**

```
<html lang="en">
<head>
	<meta charset="UTF-8"/>
	<title>Iniciación DOM 1</title>
</head>
<body>
	<form name="formulario">
		Texto: <input type="text" name="texto"/>
		<input type="button" name="crear" value="Crear">
	</form>
</body>
<script type="text/javascript">
	function crear() {
		let texto = document.formulario.texto.value;
		console.log(texto);

		//Creamos la etiqueta
		let parrafo = document.createElement("p");
		let nodoTexto = document.createTextNode(texto);

		let span = document.createElement("span");
		let nodoTexto2 = document.createTextNode("Prueba");
		span.appendChild(nodoTexto2);

		parrafo.appendChild(nodoTexto);
		document.body.appendChild(parrafo);
		let devuelto = document.body.replaceChild(span, parrafo);
		console.log(devuelto);
	}

	window.onload = function () {
		document.formulario.crear.addEventListener("click", crear);
	}
</script>
</html>
```

2. **Ejercicio 2: Crea una aplicación web con dos botones: crear y destruir. Cuando el usuario pulse "crear", se añade al body una nueva lista <ul>, cuando pulsa "destruir", se elimina la primera lista que se creó.**