UD4 – BOM y DOM

1. Introducción

Hasta ahora hemos aprendido a programar con JavaScript usando estructuras de control, variables, funciones, arrays y objetos. Todo eso nos permite procesar información, pero aún no hemos tocado la parte visible de una página web.

Con esta unidad damos el salto a lo que realmente hace que JavaScript sea el lenguaje del navegador: la manipulación del entorno web y de los documentos HTML.

En este tema vamos a conocer dos pilares fundamentales:

- El **BOM (Browser Object Model)**, que nos permite acceder y controlar el navegador.
- El **DOM (Document Object Model)**, que nos permite acceder y modificar el contenido del documento HTML.

Gracias a ambos modelos podremos crear comportamientos interactivos: abrir o cerrar ventanas, modificar el contenido de un párrafo, cambiar estilos desde JavaScript o responder a eventos como un clic.

JavaScript frente a otros lenguajes

A diferencia de lenguajes como Python, C# o Java, JavaScript tiene acceso directo al navegador sin instalar nada adicional. Mientras que en otros lenguajes hay que usar librerías externas para automatizar un navegador (como *Selenium* en Python o *JSoup* en Java), JavaScript puede hacerlo de forma nativa, porque está diseñado para ejecutarse dentro del propio navegador.

Así, mientras Python o Java trabajan sobre el lado del servidor o de escritorio, JavaScript trabaja directamente sobre el lado del cliente, controlando la interfaz y la interacción del usuario.

2. Conceptos clave y fundamentos

2.1. El BOM (Browser Object Model)

El BOM está formado por un conjunto de objetos que permiten manipular el entorno del navegador: la ventana, la URL, el historial, la pantalla, etc.

Los principales objetos son:

- `window`
- `location`

- `history`
- `navigator`
- `screen`

Veamos cada uno.

El objeto `window`

`window` es el **objeto global** de JavaScript en el navegador. Todo lo que declaramos (variables, funciones, objetos) pertenece implícitamente a `window`.

```
alert("Hola");            // equivalente a window.alert("Hola")

console.log(innerWidth);  // equivalente a window.innerWidth
```

Propiedades y métodos principales:

Propiedad / Método	Descripción
`innerWidth`, `innerHeight`	Tamaño visible de la ventana (zona de contenido).
`outerWidth`, `outerHeight`	Tamaño total de la ventana, incluyendo bordes y barras.
`open(url, nombre, especificaciones)`	Abre una nueva ventana o pestaña.
`close()`	Cierra la ventana actual (si fue abierta por el script).
`alert()`, `confirm()`, `prompt()`	Muestran cuadros de diálogo.
`print()`	Abre el cuadro de impresión.
`setTimeout()`, `setInterval()`	Ejecutan funciones con retardo o repetición.

Ejemplo: abrir y cerrar una ventana

```
let ventana = window.open("https://www.mozilla.org", "mozilla", "width=600,height=400");

setTimeout(() => ventana.close(), 3000);
```

Nota: Los navegadores modernos bloquean las ventanas emergentes que no se abren tras una acción del usuario (como un clic).

El objeto `location`

Proporciona información sobre la URL actual y permite redirigir o recargar la página.

```
console.log(location.href);     // URL completa
console.log(location.hostname); // Dominio (www.wikipedia.org)
```

Propiedades más usadas:

Propiedad	Descripción
`href`	URL completa de la página actual.
`protocol`	Protocolo (http:, https:).
`hostname`	Nombre del dominio.
`pathname`	Ruta del archivo.
`port`	Puerto utilizado.

Métodos:

Método	Descripción
`assign(url)`	Carga una nueva URL (se guarda en el historial).
`replace(url)`	Carga una nueva URL sin guardar en el historial.
`reload()`	Recarga la página actual.

Ejemplo: redirigir con confirmación

```
if (confirm("¿Quieres ir a wikipedia?")) {
  location.href = "https://www.wikipedia.org";
}
```

Error común: usar `replace()` cuando se pretende conservar el historial.
`replace()` sustituye la URL actual; no permite volver atrás.

El objeto `history`

Permite navegar por el historial del navegador.

Métodos:

Método	Descripción
`back()`	Retrocede una página.
`forward()`	Avanza una página.
`go(n)`	Avanza o retrocede *n* pasos.
`length`	Devuelve el número de entradas del historial.

Ejemplo

```
document.getElementById("volver").addEventListener("click", () => {
  history.back();
});
```

El objeto `navigator`

Proporciona información sobre el navegador y el sistema del usuario.

Propiedades comunes:

Propiedad	Descripción
`userAgent`	Devuelve una cadena con el tipo y versión del navegador.
`language`	Idioma predeterminado.
`platform`	Sistema operativo.
`onLine`	Devuelve true o false según haya conexión.

Ejemplo

```
console.log("Navegador: " + navigator.userAgent);
console.log("Idioma: " + navigator.language);
```

Nota: nunca se deben tomar decisiones críticas basadas en `userAgent`, ya que puede falsificarse.

El objeto `screen`

Proporciona información sobre la pantalla del dispositivo.

Propiedades:

Propiedad	Descripción
`width`, `height`	Resolución total.
`availWidth`, `availHeight`	Área disponible (sin la barra de tareas).
`orientation.type`	Orientación (portrait o landscape).

Ejemplo

```
console.log(`Resolución: ${screen.width}x${screen.height}`);
```

2.2. El DOM (Document Object Model)

El DOM es una representación en forma de árbol de todos los elementos HTML de la página. Cada etiqueta, atributo o texto es un nodo, y JavaScript puede acceder a ellos para leer, modificar o eliminar su contenido.

Selección de elementos

Método	Descripción
`getElementById(id)`	Devuelve el elemento con ese id.
`getElementsByClassName(clase)`	Devuelve todos los elementos con esa clase.
`getElementsByTagName(etiqueta)`	Devuelve todos los elementos con esa etiqueta.
`querySelector(selectorCSS)`	Devuelve el primer elemento que cumpla el selector.
`querySelectorAll(selectorCSS)`	Devuelve todos los elementos que cumplan el selector.

Ejemplo

```
const titulo = document.getElementById("titulo");

titulo.style.color = "darkred";
```

Nota:
`querySelectorAll()` devuelve un *NodeList* (similar a un array), pero no tiene todos sus métodos.

Sin embargo, podemos convertirlo así en array para trabajar con los valores si lo necesitamos:

```
let nodos = Array.from(document.querySelectorAll("p"));
```

Crear, modificar y eliminar elementos

```
const nuevo = document.createElement("p");

nuevo.textContent = "Este párrafo se ha creado con JavaScript";

document.body.appendChild(nuevo);
```

Otros métodos útiles:

- `removeChild(elemento)`
- `replaceChild(nuevo, antiguo)`
- `insertBefore(nuevo, referencia)`

Acceso y modificación de atributos y clases

```
const enlace = document.querySelector("a");

enlace.setAttribute("target", "_blank");


enlace.classList.add("activo");

enlace.classList.toggle("oculto");
```

Error común: usar `innerHTML` sin validar contenido. Si incluye código HTML, puede generar vulnerabilidades o errores de renderizado.

3. Ejemplos de código y casos reales

1. **Mostrar dimensiones de la ventana**

```
window.addEventListener("resize", () => {
  console.log(`Ancho: ${innerWidth}px, Alto: ${innerHeight}px`);
});
```

2. **Cambiar color al hacer clic**

```
<h2 id="texto">Haz clic aquí</h2>

<script>
```

```
document.getElementById("texto").onclick = function(){
  this.style.color = "green";
};
</script>
```

3. **Añadir elementos dinámicamente**

```
const lista = document.getElementById("tareas");
const item = document.createElement("li");
item.textContent = "Aprender DOM";
lista.appendChild(item);
```

4. Errores comunes y buenas prácticas

- Llamar a `document.write()` después de cargar la página → borra todo el HTML.
- Intentar acceder a un elemento antes de que se cargue el DOM.
  Solución: envolver el código en `window.onload = function() { ... }`.
- Confundir `innerText` con `innerHTML`.
- Manipular el DOM dentro de bucles sin optimizar → ralentiza la página.

5. Curiosidades o ampliación

El DOM no pertenece a JavaScript; es un **estándar del W3C** que también utilizan otros lenguajes. Por ejemplo, Python puede manipular DOM con *BeautifulSoup* o *Selenium*, pero necesita controladores externos, JavaScript no necesita controladores externos.

`window` es exclusivo del navegador; en Node.js se usa `global`.

6. Ejercicios prácticos y autoevaluación

1. Crea un botón que abra una nueva ventana y la cierre automáticamente a los 5 segundos.
2. Muestra un mensaje de alerta con el tamaño de la ventana actual.
3. Crea una lista de elementos con `createElement()` y añade un botón para eliminar el último.
4. Crea un enlace que pregunte con `confirm()` si el usuario realmente quiere abandonar la página.
5. Cambia el color del texto de un párrafo cuando pases el ratón por encima y recupéralo al salir.

7. Cuadro de referencia rápida

Objeto	Propiedades más usadas	Métodos más usados
window	innerWidth, innerHeight, outerWidth, outerHeight	alert(), prompt(), confirm(), open(), close(), print(), setTimeout(), setInterval()
location	href, hostname, pathname, protocol, port	assign(), replace(), reload()
history	length	back(), forward(), go()
navigator	userAgent, language, platform, onLine	(no suele tener métodos directos)
screen	width, height, availWidth, availHeight, orientation.type	—
document	title, body, forms, images, links	getElementById(), querySelector(), createElement(), appendChild(), removeChild(), setAttribute(), classList.add(), classList.toggle()