Aplicaciones Web - Práctica 4: JavaScript Aplicado al Frontend

El DOM

El Document Object Model (DOM) es una representación de la pagina web. Permite a los lenguajes de programación, como JavaScript, interactuar con la pagina web, permitiendo modificar su estructura, estilo y contenido de manera dinámica.

Formas de incluir JavaScript en HTML

Existen varias formas de incluir JavaScript en un documento HTML: 1. **Dentro de una etiqueta `<script>`**: `html <script> // Código JavaScript aquí console.log("Hola, mundo!"); </script>`

2. **Eventos en línea**: `html <button onclick="alert('¡Hola!')">Haz clic aquí</button>`
3. **Archivo externo**: `html <script src="ruta/al/archivo.js" defer></script>`

Normalmente usaremos archivos externos para incluir JavaScript en nuestras páginas web. Ya que de esta forma mantenemos el código HTML y JavaScript separados, lo que facilita el mantenimiento y la legibilidad del código.

Es habitual colocar las etiquetas `<script>` justo antes de la etiqueta de cierre `</body>` para asegurar que el DOM esté completamente cargado antes de que se ejecute el JavaScript. Una opción mejor es usar el atributo `defer` en la etiqueta `<script>`, lo que permite que el navegador descargue el archivo JavaScript mientras sigue procesando el HTML, y ejecuta el script una vez que el DOM esté completamente cargado.

```
<script src="ruta/al/archivo.js" defer></script>
```

El objeto document y selectores

El objeto `document` es la forma principal para interactuar con el DOM en JavaScript.

Los metodos más comunes para seleccionar elementos del DOM son los llamados “selectores”, los cuales sirven para obtener referencias a los elementos HTML que queremos manipular. Los selectores más importantes son: - `getElementById(id)`: Selecciona un elemento por su ID. - `getElementsByClassName(className)`: Selecciona todos los elementos que tienen una clase específica. - `getElementsByTagName(tagName)`: Selecciona todos los elementos con una etiqueta específica. - `querySelector(selector)`: Selecciona el primer elemento que coincide con un selector CSS. - `querySelectorAll(selector)`: Selecciona todos los elementos que coinciden con un selector CSS.

Manipulación de atributos y contenido

Una vez que hemos seleccionado un elemento del DOM, podemos manipular sus atributos y contenido utilizando JavaScript.

Para manipular atributos, podemos usar los métodos `getAttribute`, `setAttribute`, y `removeAttribute`. - `getAttribute(nombreAtributo)`: Obtiene el valor de un atributo específico. - `setAttribute(nombreAtributo, valor)`: Establece el valor de un atributo específico. - `removeAttribute(nombreAtributo)`: Elimina un atributo específico.

Para manipular el contenido de un elemento, podemos usar las propiedades `innerHTML`, `textContent`, y `innerText`. - `innerHTML`: Permite obtener o establecer el contenido HTML de un elemento. - `textContent`: Permite obtener o establecer el contenido de texto de un elemento, ignorando las etiquetas HTML. - `innerText`: Similar a `textContent`, pero tiene en cuenta el estilo CSS y no incluye el texto oculto.

Manipulación de estilos y clases

Una vez que hemos seleccionado un elemento del DOM, podemos manipular sus estilos y clases utilizando las propiedades `style` y `classList` del elemento. Por ejemplo:

- Para cambiar el estilo de un elemento: `javascript const elemento = document.getElementById("miElemento"); elemento.style.color = "red"; // Cambia el color del texto a rojo elemento.style.backgroundColor = "yellow"; // Cambia el color de fondo a amarillo`

- Para agregar, eliminar o alternar clases: `javascript const elemento = document.getElementById("miElemento"); elemento.classList.add("nuevaClase"); // Agrega una clase elemento.classList.remove("claseExistente" // Elimina una clase elemento.classList.toggle("otraClase"); // Alterna una clase, si la tiene la elimina, si no la tiene la agrega`

Creación y eliminación de elementos del DOM

Una vez que hemos seleccionado un elemento del DOM, podemos crear, eliminar o modificar elementos utilizando métodos como `createElement`, `appendChild` y `removeChild`. Por ejemplo:

- Para crear y agregar un nuevo elemento:

```
const nuevoElemento = document.createElement("div");
nuevoElemento.textContent = "¡Hola, mundo!";
document.body.appendChild(nuevoElemento);
```

- Para eliminar un elemento:

```
const elementoAEliminar = document.getElementById("miElemento");
elementoAEliminar.parentNode.removeChild(elementoAEliminar);
```

Eventos y Listeners

Los eventos son acciones que ocurren en el navegador, como clics de botones, movimientos del ratón, o envíos de formularios. Los listeners son funciones que se ejecutan en respuesta a estos eventos. Se pueden agregar listeners utilizando el método `addEventListener`. Por ejemplo:

```
const boton = document.getElementById("miBoton"); // Selecciona el botón por su ID

boton.addEventListener("click", miFuncion); // Agrega un listener para el evento 'click'

function miFuncion() {
    alert("¡Botón clickeado!");
}
```

En este ejemplo, cuando el usuario hace clic en el botón con el ID `miBoton`, se ejecuta la función `miFuncion`, que muestra una alerta.

Algunos de los eventos más comunes son: - `click`: Se activa cuando se hace clic en el elemento. - `mouseover`: Se activa cuando el puntero del ratón se mueve sobre el elemento. - `mouseout`: Se activa cuando el puntero del ratón se mueve fuera del elemento. - `submit`: Se activa cuando se envía un formulario. Solo se aplica a elementos `<form>`. - `input`: Se activa cuando el valor de un campo de entrada cambia. Solo se aplica a elementos `<input>`, `<textarea>`, y `<select>`. - `change`: Se activa cuando el valor de un campo de entrada pierde el foco y ha cambiado. - `focus`: Se activa cuando un elemento recibe el foco. - `blur`: Se activa cuando un elemento pierde el foco. - `DOMContentLoaded`: Se activa cuando el DOM ha sido completamente cargado. Solo se aplica al objeto `document`. Preferible usar el atributo `defer` en la etiqueta `<script>` para este propósito.

Formularios