El contexto global u objeto padre en JavaScript de los navegadores es **window**, de window cuelgan todas las APIs del navegador.

Document.scripts 🡪 nos trae los scripts que están linkeados en el html.

Texto

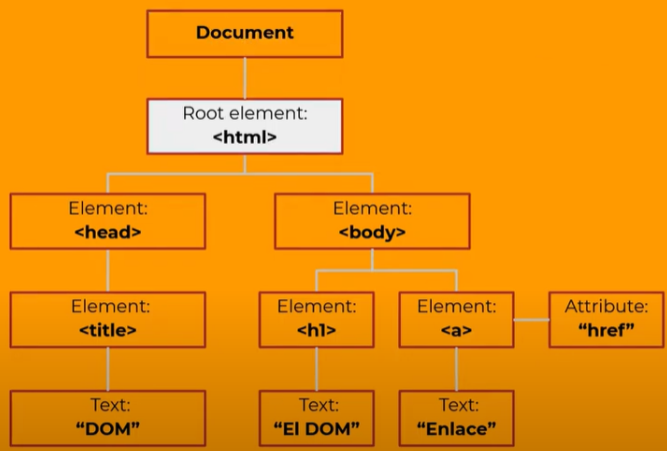
Descripción generada automáticamente

**¿Qué es el DOM?**

El DOM (Document Object Model) es una representación estructurada en forma de Árbol de un documento HTML o XML. Nos permite interactuar y manipular la estructura, contenido, estilo, reaccionar eventos o a crearlos del documento HTML de manera dinámica. Cada parte que compone el documento es un nodo dentro del árbol.

El DOM es una API que ayuda a los desarrolladores a interactuar con los nodos de documentos dentro de un navegador.

**Estructura del DOM**



* **Nodo raíz:** El nodo más alto del árbol es el documento (**document**), que es el punto de entrada para interactuar con el contenido.
* **Nodos de elemento:** Representan las etiquetas HTML, como **<html>, <body>,** **<div>**, **<p>**, etc. Cada nodo de elemento puede tener **atributos**, **hijos** y **texto interno**.
* **Nodos de texto:** Cada nodo de elemento puede contener un nodo de texto que representa el contenido textual entre las etiquetas. El texto dentro de un párrafo sería un nodo de texto por ejemplo.
* **Nodos de atributos:** Los atributos de los elementos como **id**, **class**, **src**, etc; también son nodos, aunque no forman parte directa de la estructura de árbol, sino que están asociados con los nodos de elemento correspondiente.
* **Nodos de comentario:** Los comentarios en HTML **<!--Comentario -->** se almacenan en nodos de comentario.

**Seleccionar elementos del DOM:** El objeto `**document**` en el DOM es un objeto central que representa todo el documento HTML actual en un navegador web. Es un punto de entrada esencial para interactuar con la estructura y el contenido de una página web desde JavaScript.

1. **Por ID:** Devuelve el elemento con el ID especificado, toma como argumento un string con el nombre del id.

**Retorno:** El elemento del DOM o null si no lo encuentra.

****

1. **Por nombre de la clase:** Devuelve una colección de tipo **HTML collection** de elementos con la clase especificada, toma como argumento un string con el nombre de clase. El HTML Collection es como los arreglos pero sin algunos de sus métodos.

**Retorno:** HTML Collection de elementos.

****

1. **Por nombre de la etiqueta:** Devuelve una colección de elementos con el nombre de etiqueta especificado, toma como argumento un string del nombre de la etiqueta.

**Retorno:** HTML Collection de elementos.

****

1. **Por selector CSS:** Devuelve el primer elemento que coincida con el selector CSS proporcionado, el selector puede ser cualquier selector válido de CSS (clases, IDs, combinaciones, etc.).

**Retorno:** Devuelve el primer elemento que coincide con el selector, o null si no encuentra coincidencias.

****

1. **Todos los elementos por selector CSS:** Devuelve todos los elementos que coincidan con el selector CSS proporcionado.

**Retorno:** Devuelve una NodeList con todos los elementos que coinciden con el selector. La NodeList no es una lista viva como la HTMLCollection, pero puede iterarse con métodos como forEach.

****

1. **Por name:** Devuelve todos los elementos que tienen un atributo name específico (generalmente usado en formularios).

**Retorno:** Devuelve una NodeList con todos los elementos que coinciden con el valor del atributo name.

**Texto

Descripción generada automáticamente**

1. **Por un elemento padre:** Devuelve el primer elemento que coincide con el selector dentro del nodo padre.

**Retorno:** Devuelve el primer elemento que coincide con el selector dentro del nodo padre, o null si no encuentra coincidencias.

**parentElement.querySelector(selector)**

**Texto

Descripción generada automáticamente**

1. **Todos los elementos por un elemento padre:** Devuelve todos los elementos que coinciden con el selector dentro del nodo padre.

**Retorno:** Devuelve una **NodeList** con todos los elementos que coinciden con el selector dentro del contexto del elemento padre.

**parentElement.querySelectorAll(selector)**

**Texto

Descripción generada automáticamente**

**Modificar elementos HTML**

1. **innerHTML:** Permite obtener o modificar el contenido HTML de un elemento del DOM. Si se establece un nuevo valor, reemplaza todo el contenido actual del elemento. Si no se controla adecuadamente, puede ser una puerta para ataques de inyección de código.

**Texto

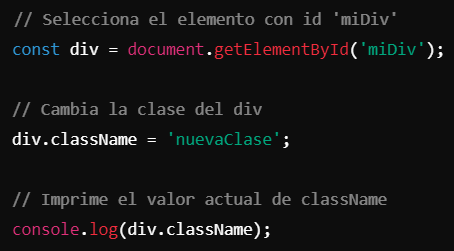
Descripción generada automáticamente**

1. **textContent:** Obtiene o establece el contenido de texto de un elemento, ignorando el HTML. Es útil para evitar que el contenido HTML sea interpretado o inyectado.

**Texto

Descripción generada automáticamente**

1. **className:** Permite obtener o cambiar el valor del atributo class de un elemento. Reemplaza todas las clases actuales del elemento por las que se especifiquen.

****

1. **classList:** Ofrece una manera más flexible de manipular las clases de un elemento, permitiendo añadir, eliminar o alternar clases sin afectar otras.

**Texto

Descripción generada automáticamente**

1. **value:** Se utiliza para obtener o establecer el valor de los elementos de formulario, como <input>, <textarea>, o <select>.

**Texto

Descripción generada automáticamente**

1. **style:** Permite modificar los estilos en línea de un elemento. Cada propiedad de estilo CSS se convierte en una propiedad de este objeto.

**Texto

Descripción generada automáticamente**

**Atributos de document**

1. **document.title:** Este atributo permite acceder y modificar el título de la página web, que es lo que aparece en la pestaña del navegador.

****

1. **document.body:** Proporciona acceso directo al elemento <body> del documento HTML, que contiene el contenido principal de la página.

**Texto

Descripción generada automáticamente**

1. **document.cookie:** Permite acceder, leer y escribir cookies en el navegador. Las cookies son pequeñas piezas de datos que se almacenan en el navegador y se envían junto con cada solicitud HTTP.

**Texto

Descripción generada automáticamente**

1. **document.documentElement:** Hace referencia al elemento raíz del documento, que es el elemento <html>. Esto te da acceso directo a todo el contenido de la página.

**Texto

Descripción generada automáticamente**

1. **document.head:** Proporciona acceso al elemento <head>, que es donde se suelen incluir metadatos, enlaces a hojas de estilo, scripts y otros elementos importantes para la configuración de la página.

**Texto

Descripción generada automáticamente**

**El traversing del DOM:** Es un término que se utiliza en programación web para referirse a la acción de recorrer y navegar por la estructura jerárquica de un documento HTML utilizando JavaScript. Es decir, implica moverse entre los nodos en la estructura del árbol.

1. **parentNode:** Accede al nodo padre de un elemento. Retorna el nodo padre inmediato.

Texto

Descripción generada automáticamente

1. **children:** Devuelve una colección de los elementos hijos del nodo actual. Solo incluye nodos de tipo elemento (no texto o comentarios).

Texto

Descripción generada automáticamente

1. **firstChild:** Devuelve el primer hijo del nodo, que puede ser un nodo de texto, comentario o elemento.

**Texto

Descripción generada automáticamente**

1. **lastChild:** Devuelve el último hijo del nodo, que puede ser un nodo de texto, comentario o elemento.

**Texto

Descripción generada automáticamente**

1. **nextSibling:** Devuelve el siguiente nodo hermano en el DOM, que puede ser un nodo de texto, comentario o elemento.

**Texto

Descripción generada automáticamente**

1. **previousSibling:** Devuelve el nodo hermano anterior en el DOM, que puede ser un nodo de texto, comentario o elemento.

Texto

Descripción generada automáticamente

1. **parentElement:** Accede al elemento padre del nodo actual. Similar a parentNode, pero garantiza que el nodo padre sea un elemento, no un nodo de texto o comentario.

Texto

Descripción generada automáticamente

1. **appendChild:** Se utiliza para agregar un nodo (elemento) como hijo de otro nodo. Generalmente, se usa para agregar un elemento HTML a otro, como agregar un párrafo a un div o un elemento de lista a una lista desordenada (ul)

Texto

Descripción generada automáticamente

1. **removeChild:** Se utiliza para eliminar un nodo hijo de un nodo padre. Se pasa como argumento el nodo hijo que se desea eliminar.Texto

   Descripción generada automáticamente

**Métodos para crear y modificar elementos**

1. **document.createElement(tagName):** Crea un nuevo elemento con el nombre de etiqueta especificado.

**Texto

Descripción generada automáticamente**

1. **element.setAttribute(name, value):** Establece un atributo en el elemento con el nombre y el valor especificado.

**Texto

Descripción generada automáticamente**

1. **element.removeAttribute(name):** Elimina el atributo especificado del elemento.

**Texto

Descripción generada automáticamente**

1. **element.classList.add(className):** Añade una o más clases al elemento.

**Texto

Descripción generada automáticamente**

1. **element.classList.remove(className):** Elimina una o más clases del elemento.

**Texto

Descripción generada automáticamente**

**Método insertAdjacentElement(position, element):** Es una función del DOM que permite insertar un elemento en relación con un elemento existente en el documento.

****

**element:** El elemento que deseas insertar en el DOM.

**position:** Una cadena que indica la posición en la que deseas insertar el nuevo elemento. Puede ser uno de los siguientes valores:

**'beforebegin':** Inserta el nuevo elemento justo antes del elemento objetivo (es decir, antes del inicio del elemento).

**'afterbegin':** Inserta el nuevo elemento justo dentro del elemento objetivo, como su primer hijo.

**'beforeend':** Inserta el nuevo elemento justo dentro del elemento objetivo, como su último hijo.

**'afterend':** Inserta el nuevo elemento justo después del elemento objetivo (es decir, después del cierre del elemento).

**Objeto event:** Es un objeto que representa un evento que ha ocurrido en la página web. Este objeto contiene información detallada sobre el evento, como el tipo de evento, el elemento DOM que desencadenó el evento, las coordenadas del evento en la pantalla, las teclas que se presionaron (en el caso de eventos de teclado), entre otros datos relevantes. El objeto "event" se pasa como un parámetro a la función de manejo de eventos cuando se produce un evento. A través de este objeto, puedes acceder a información específica sobre el evento y utilizarla para tomar decisiones o realizar acciones en respuesta al evento.

Texto

Descripción generada automáticamente

**Eventos del DOM:** Es una acción u ocurrencia que sucede en el navegador web y que puede ser detectada y manejada por el código JavaScript. Estos eventos pueden ser causados por la interacción del usuario con la página web, como hacer clic en un botón, mover el ratón, escribir en un campo de entrada de texto, entre otros, o pueden ser generados por el propio navegador, como la carga completa de una página.

**Sintaxis de la función AddEventListener():** La función addEventListener se utiliza para registrar un manejador de eventos en un elemento del DOM. Esto significa que cuando ocurre un evento específico en el elemento, el manejador (una función) se ejecuta.

****

* **eventType:** El tipo de evento que quieres escuchar. Es una cadena de texto que indica el nombre del evento (por ejemplo, **'click'**, **'mouseover'**, **'keydown'**, etc.).
* **listener:** La función que se ejecutará cuando el evento ocurra. Esta función recibe un objeto event que contiene información sobre el evento.
* **options (opcional):** Un objeto o un booleano que especifica características adicionales sobre el evento.

**Eventos del mouse**

1. **click:** Ocurre cuando se hace clic en un elemento del HTML.

**Texto

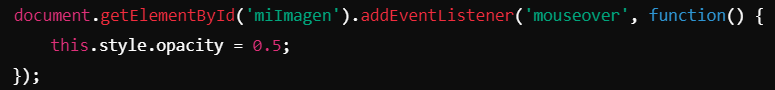
Descripción generada automáticamente**

1. **dblclick:** Ocurre cuando se hace doble clic en un elemento.

**Texto

Descripción generada automáticamente**

1. **mouseover**: Ocurre cuando el puntero del mouse se coloca sobre un elemento.



**Eventos del teclado**

1. **keydown:** Ocurre cuando se presiona una tecla.

Texto

Descripción generada automáticamente

1. **keyup**: Ocurre cuando se suelta una tecla.

Texto

Descripción generada automáticamente

1. **keypress**: Ocurre cuando se presiona una tecla (en desuso en algunas implementaciones modernas).

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

**Eventos del formulario**

1. **submit:** Ocurre cuando se envía un formulario.

**Texto

Descripción generada automáticamente**

1. **change:** Ocurre cuando el valor de un elemento de formulario cambia.

**Texto

Descripción generada automáticamente**

**Eventos de la ventana**

1. **load:** Ocurre cuando la ventana o un recurso se ha cargado completamente.

**Texto

Descripción generada automáticamente**

1. **resize:** Ocurre cuando se redimensiona la ventana.

**Texto

Descripción generada automáticamente**

1. **scroll:** Ocurre cuando se desplaza la ventana.

**Texto

Descripción generada automáticamente**

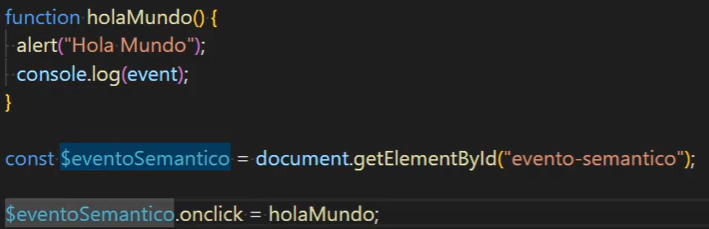
**Existen 3 formas distintas de controlar eventos**

* **Desde el HTML:** Puedes agregar manejadores de eventos directamente en el HTML usando atributos. Este método es simple pero no es el más recomendado para aplicaciones más grandes, ya que mezcla la lógica con la estructura del HTML. El atributo **ON (**Hace referencia a eventos**)**, luego lo que se le pasa entre comillas es el script que va a ejecutar cuando suceda el evento.

****

* **Evento con manejador semántico:** Con este método, primero debemos seleccionar desde el JS el elemento al que queremos agregarle un evento, luego seleccionar el evento y lo que va a realizar.

Los manejadores semánticos poseen una limitante, por cada evento solo podemos asignarle una función (ya sea anónima, de flecha, etc). Tampoco puede recibir parámetros, el único que puede recibir es el evento en si con la palabra **event**

****

* **Evento con manejadores múltiples:** Es el uso de **addEventListener.** La ventaja de utilizarlo de esta forma es que un elemento puede tener más de un evento asociado y no habrá problema alguno.

Siempre que trabajemos con eventos, tenemos el objeto **event** que nos proporcionará información acerca del evento.

Si hacemos **console.log(event)** 🡪 nos aparecerá un objeto de tipo event, el cual tendrá toda la información del evento.

**Type:** Es el tipo de evento que se desencadenó.

**Target:** El elemento que originó el evento, desde target tenemos acceso a todas las propiedades de ese elemento HTML

**Concepto de un programa en ejecución**

* **Stack (Pila):** Almacena información sobre las llamadas a funciones y variables locales. Las funciones se apilan y desapilan en este orden cuando se llaman y completan.
* **Frame (Marco):** Cada vez que se llama a una función, se crea un nuevo marco en la pila. Este marco contiene variables locales y la dirección de retorno.
* **Heap (Montículo):** Almacena objetos y variables dinámicas. A diferencia del stack, el heap no tiene un tamaño fijo, y la memoria es gestionada por el recolector de basura de JavaScript.
* **Queue (Cola):** Maneja la ejecución de eventos y tareas pendientes. Los eventos y tareas se encolan y se procesan en orden.

**Event Loop:** El Event Loop es el mecanismo que permite a JavaScript ejecutar código, recolectar y procesar eventos, y ejecutar sub-tareas en un orden específico.

El Event Loop revisa la cola de tareas y ejecuta las tareas pendientes una vez que la pila de llamadas está vacía.

* **Ejecutar código sincronizado.**
* **Procesar eventos en la cola de eventos.**
* **Ejecutar micro-tareas (como Promise y MutationObserver).**
* **Repita el ciclo.**