
# El DOM (Document Object Model)

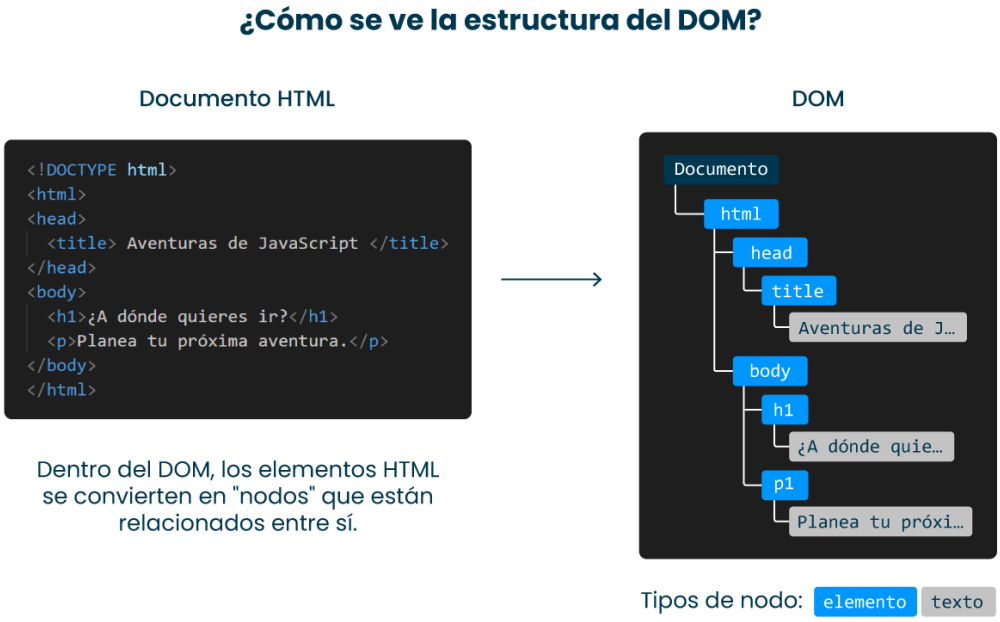
El DOM, o Modelo de Objetos del Documento (en inglés, Document Object Model), es una interfaz de programación para documentos HTML y XML. Esencialmente, el DOM representa la estructura de un documento como un árbol de nodos, donde cada nodo corresponde a un elemento, atributo o contenido de texto en el documento.

El DOM proporciona una forma de representar la estructura de un documento de manera que los programas puedan cambiar la estructura, estilo y contenido del documento de manera dinámica. En el contexto de desarrollo web, el DOM es crucial para la interactividad de las páginas web, ya que permite a los desarrolladores acceder, modificar y manipular los elementos y contenido de una página web mediante el uso de lenguajes de programación como JavaScript.

## ¿Qué es un nodo?

El DOM se construye como un árbol de objetos. Es decir, que **todos los elementos de HTML son definidos como objetos.**

Un nodo es cualquier etiqueta que se encuentra en el **html.**



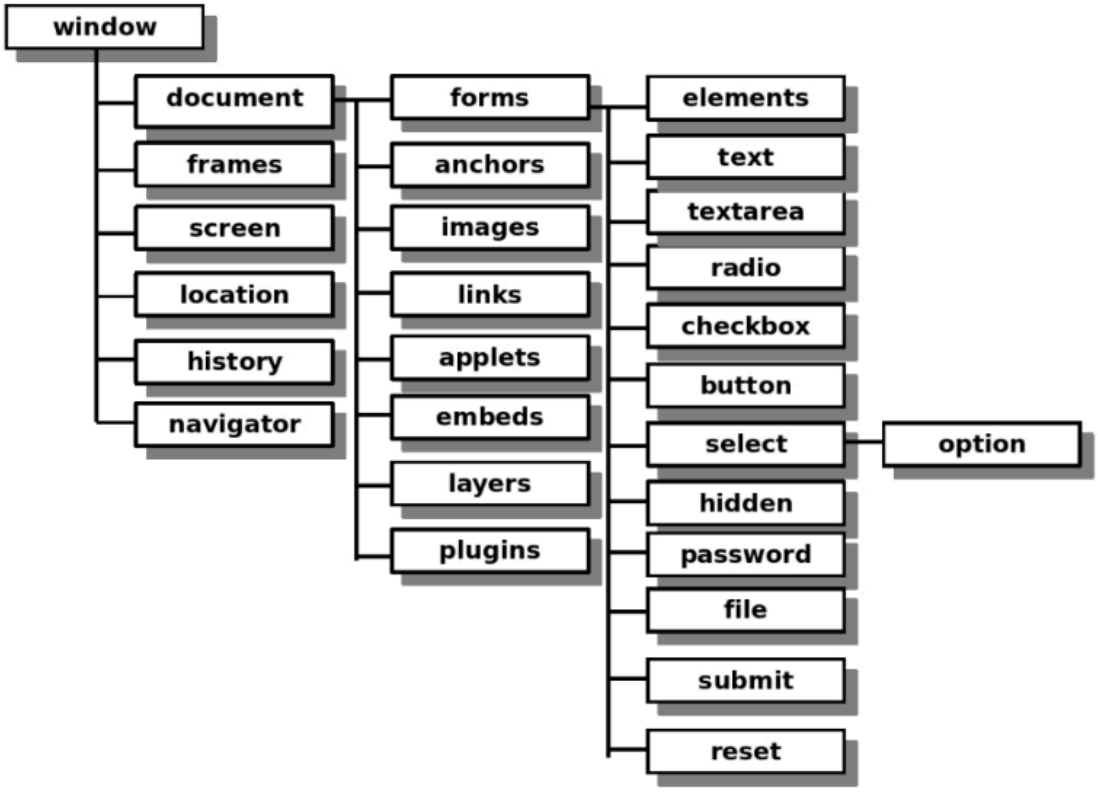

Es una representación similar a un árbol del contenido de una página web, es decir, un árbol de "nodos" con diferentes relaciones dependiendo de cómo estén organizados en el documento HTML.

```
<div id="container">
  <div class="display"></div>
  <div class="controls"></div>
</div>
```

En el ejemplo anterior, el  es un "**hijo**" de  y un **hermano** de .

Imagínalo como un árbol genealógico.  es un padre, con sus hijos en el siguiente nivel, cada uno en su propia "rama".

En el DOM existe una jerarquía de objetos que se puede representar de la siguiente manera




## Métodos de selección de elementos

Los **métodos** son acciones que podemos realizar sobre los elementos/objetos. Un ejemplo de estas acciones es **"seleccionar los elementos".**

💡 ***Recordemos*** *que los métodos se aplican a objetos. Y los métodos que vamos a ver a continuación se aplican al objeto* ***document***

- **document.getElementById("id") -** Seleccionar un elemento a través del ID
- **document.getElementsByTagName("tag") -** Selecciona todos los elementos que coincidan con el nombre de la etiqueta especificada
- **document.querySelector("selector") -** Devuelve el primer elemento que coincida con el grupo especificado de selectores
- **document.querySelectorAll("selector") -** Devuelve todos los elementos que coincidan con el grupo especificado de selectores.

## ¿Cómo seleccionamos un elemento?

Les dejamos el siguiente video para ver cómo seleccionar elementos: Métodos de selección de elementos | JavaScript II | Egg

Es decir que para seleccionar un elemento del DOM, podemos hacer a través de los selectores de CSS, por ejemplo, dado el siguiente html:

```
<div id="container">
  <div class="display"></div>
  <div class="controls"></div>
</div>
```

Los selectores que podemos tener son:

- div.display
- .display
- #container > .display
- div#container > div.display

```
const container = document.querySelector('#container');
// Selecciona el div #container

console.dir(container.firstElementChild);
// Selecciona el primer hijo del elemento #container => que es .display

const controls = document.querySelector('.controls');
// Selecciona el elemento div que tiene la clase controls

console.dir(controls.previousElementSibling);
// Selecciona el hermano anterior => en este caso, .display
```

Entonces podríamos también seleccionar un nodo, identificando su relación de parentesco con los nodos de alrededor.

## Métodos para definir, obtener, eliminar valores de atributos

- **setAttribute() -** Modifica el valor de un atributo
- **getAttribute() -** Obtiene el valor de un atributo

- **removeAttribute() -** Remueve el valor de un atributo

## setAttribute()

La sintaxis general del método setAttribute() es la siguiente:

```
elemento.setAttribute(nombreAtributo, valorAtributo);
```

Donde:

- **elemento**: Es el elemento HTML al que se le quiere agregar o modificar un atributo.
- **nombreAtributo**: Es una cadena que representa el nombre del atributo que se desea cambiar o agregar.
- **valorAtributo**: Es el valor que se quiere asignar al atributo especificado. Puede ser una cadena de texto o cualquier otro tipo de dato admitido por el atributo en cuestión.

Ejemplo de cómo se utiliza setAttribute() para agregar o modificar atributos:

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de setAttribute()</title>
</head>
<body>
    <div id="miDiv">Este es un div</div>

    <script>
        // Obtenemos el elemento div mediante su ID
        var miDiv = document.getElementById('miDiv');

        // Agregamos el atributo "class" con el valor "destacado"
        miDiv.setAttribute('class', 'destacado');

        // Modificamos el atributo "id" para cambiar su valor
        miDiv.setAttribute('id', 'nuevoId');
    </script>
</body>
</html>
```

En este ejemplo, hemos utilizado **setAttribute()** para agregar una clase **(class="destacado")** y modificar el ID **(id="nuevoId")** del elemento div con el ID "miDiv". Después de la ejecución del script, el elemento div se verá así:

```
<div id="nuevoId" class="destacado">Este es un div</div>
```

En resumen, **setAttribute()** es una función esencial para modificar dinámicamente el contenido y comportamiento de elementos HTML en una página web utilizando JavaScript.

## getAttribute()

La sintaxis general del método **getAttribute()** es la siguiente:

```
var valorAtributo = elemento.getAttribute(nombreAtributo);
```

Donde:

- **elemento**: Es el elemento HTML del cual se quiere obtener el valor del atributo.
- **nombreAtributo**: Es una cadena que representa el nombre del atributo que se desea obtener.

Ejemplo de cómo se utiliza **getAttribute**():

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de getAttribute()</title>
</head>
<body>
    <div id="miDiv" class="destacado">Este es un div</div>

    <script>
        // Obtenemos el elemento div mediante su ID
        var miDiv = document.getElementById('miDiv');

        // Obtenemos el valor del atributo "class"
        var claseDelDiv = miDiv.getAttribute('class');
        console.log(claseDelDiv); // Mostrará "destacado"
```

```
    </script>
</body>
</html>
```

En este ejemplo, hemos utilizado **getAttribute**() para obtener el valor del atributo "**class**" del elemento div con el ID "**miDiv**". La variable claseDelDiv contendrá el valor "**destacado**".

Es importante mencionar que **getAttribute**() devuelve siempre el valor del atributo como una cadena de texto. Si el atributo no está presente o no tiene un valor asignado, el método devolverá **null**.

Además, si se quiere acceder a atributos especiales como **href**, **src**, **data**-*, **etc**., es posible que los navegadores modernos apliquen ciertas transformaciones automáticas al valor devuelto por **getAttribute**(), por lo que puede haber diferencias en la forma en que se obtienen estos atributos en comparación con el código fuente HTML original.

## removeAttribute()

La sintaxis general del método **removeAttribute**() es la siguiente:

```
elemento.removeAttribute(nombreAtributo);
```

Donde:

- **elemento**: Es el elemento HTML del cual se quiere eliminar el atributo.
- **nombreAtributo**: Es una cadena que representa el nombre del atributo que se desea eliminar.

Ejemplo de cómo se utiliza **removeAttribute**():

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de removeAttribute()</title>
</head>
<body>
```

```
    <div id="miDiv" class="destacado">Este es un div</div>

    <script>
        // Obtenemos el elemento div mediante su ID
        var miDiv = document.getElementById('miDiv');

        // Eliminamos el atributo "class" del elemento div
        miDiv.removeAttribute('class');
    </script>
</body>
</html>
```

En este ejemplo, hemos utilizado **removeAttribute**() para eliminar el atributo "class" del elemento div con el ID "**miDiv**". Después de la ejecución del script, el elemento div ya no tendrá el atributo "**class**".

Es importante tener en cuenta que al eliminar un atributo con **removeAttribute**(), se revertirá cualquier estilo o comportamiento asociado a ese atributo, ya que el elemento volverá a su estado predeterminado. Si el atributo que se intenta eliminar no existe en el elemento, **removeAttribute**() no provocará ningún error o excepción.

El uso de **removeAttribute**() puede ser beneficioso cuando se necesita realizar cambios en la estructura y presentación de una página web de manera dinámica en función de ciertas acciones o eventos.

💡Para más detalles sobre atributos se puede revisar la documentación de Mozilla HTML attirbute reference

## Métodos para agregar elementos

### createElement()

A través de **createElement**, podemos crear nuevos elementos HTML que luego pueden ser agregados al DOM de una página web.

En JavaScript, la sintaxis para usar **createElement** es la siguiente:

```
document.createElement(tagName)
```

Donde:

- **document** es el objeto que representa el DOM en una página web.
- **tagName** es una cadena de texto que especifica el nombre del elemento HTML que queremos crear, como "div", "p", "span", "a", etc.

Ejemplo de cómo se usa **createElement** para crear un nuevo elemento <p> (párrafo) y luego agregarlo al DOM:

```
// Creamos un nuevo elemento <p>
const newParagraph = document.createElement("p");

// Asignamos contenido al párrafo
newParagraph.textContent = "Este es un nuevo párrafo creado con
JavaScript.";

// Agregamos el párrafo al final del body del documento
document.body.appendChild(newParagraph);
```

En este ejemplo, hemos creado un nuevo **párrafo** (<p>) utilizando **createElement**, luego le hemos asignado texto utilizando la propiedad **textContent**, y finalmente, lo hemos agregado al final del elemento body del DOM utilizando el método **appendChild**.

## Métodos para alterar propiedades de elementos

### Agregar estilos en línea

```
div.style.color = 'blue';
// le agrega color de letra azul al div

div.style.cssText = 'color: blue; background: white;';
// agrega más de un estilo

div.setAttribute('style', 'color: blue; background: white;');
// le agrega varios atributos de estilo, como color y background
```

## Trabajando con clases

```
div.classList.add('new');
// agrega la clase new al div

div.classList.remove('new');
// remueve la clase new al div

div.classList.toggle('active');
// si el div tiene la clase active, la remueve, si no la tiene, la agrega
```

## Agregando o editando texto

Si el contenido del div tiene texto, podemos utilizar:

```
div.textContent = 'Hello World!'
// crea un nodo de texto que contiene la frase Hello World!
// Lo inserta en el div

div.textContent = 'Hello World!'
// modifica el texto del div por la frase Hello World!
```

## Agregando o editando valores

Si el elemento es un input y tiene núemeros, podemos utilizar:

```
input.value = 23
// modifica el valor del input por el valor 23
```

## Agregando contenido html

```
div.innerHTML = '<span>Hello World!</span>';
// renderiza el html dentro del div
```

# DOM - Eventos

Los eventos constituyen la esencia dinámica de nuestras páginas web, brindándonos la capacidad de gestionar de forma interactiva nuestro DOM. Estas acciones, que van desde clics hasta pulsaciones de teclas, se convierten en puntos de interés cruciales. A través de JavaScript, podemos dotar a nuestra página web con la habilidad de no solo detectar sino también responder a estos eventos.

Existen tres enfoques fundamentales para llevar a cabo esta gestión de eventos:

- **Atributos de función en HTML:** Puedes especificar directamente atributos de función en tus elementos HTML, lo que permite ejecutar acciones específicas al interactuar con ellos.
- **Propiedades de tipo [eventType] en JavaScript:** Al establecer propiedades como onclick o onmousedown en los nodos del DOM mediante JavaScript, puedes asignar funciones que se ejecutarán en respuesta a eventos específicos.
- **Escuchadores de eventos en JavaScript:** La opción preferida es el uso de escuchadores de eventos. Estos permiten adjuntar funciones específicas a los nodos del DOM para manejar eventos de manera más flexible y organizada.

Aunque los escuchadores de eventos son la metodología recomendada, es probable que te encuentres con los otros dos enfoques con regularidad. Exploraremos en detalle cada uno de estos métodos en los siguientes pasos para una comprensión más completa de cómo implementar y aprovechar al máximo la gestión de eventos en tus proyectos web.