Javascript y el DOM

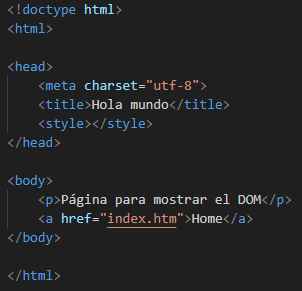
¿Qué es DOM?

El DOM es básicamente la estructura de los elementos HTML que genera el navegador a partir de las etiquetas HTML del archivo que está intentando desplegar visualmente.

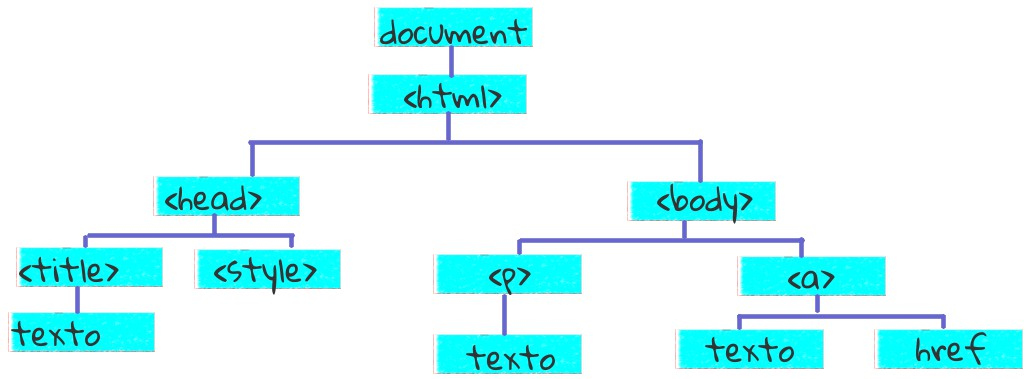
Lectura adicional: <https://www.aulascript.com/dhtml/el_dom.htm>

Ejemplo:

Para el siguiente código HTML:



El browser construye el siguiente DOM

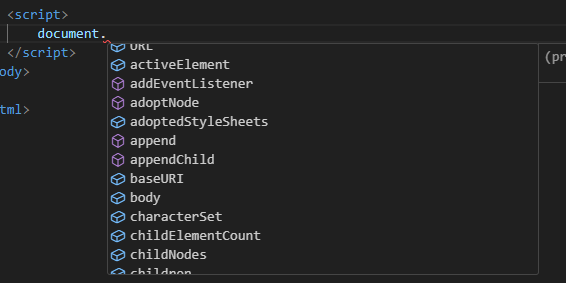


La estructura del DOM es una estructura tipo árbol, donde la etiqueta document sería la cabeza del árbol y cada uno de los elementos HTML (HTMLObjects) se denomina Nodo.

**Conceptos claves:**

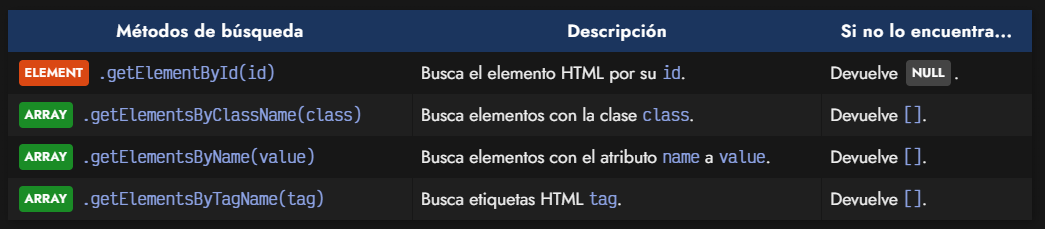
**Objeto DOM:**  es el objeto que representa al DOM accesible por Javascript. El objeto está almacenado en la variable llamada “document”.

Este objeto posee las propiedades necesarias para acceder a todos los nodos y propiedades.



En la imagen podemos ver como se puede acceder al objeto DOM.

Para acceder a los elementos del DOM podemos utilizar diferentes funciones, algunas de ellas son:

****

Para leer en mas detalle visitar: <https://lenguajejs.com/dom/buscar/getelementbyid/>

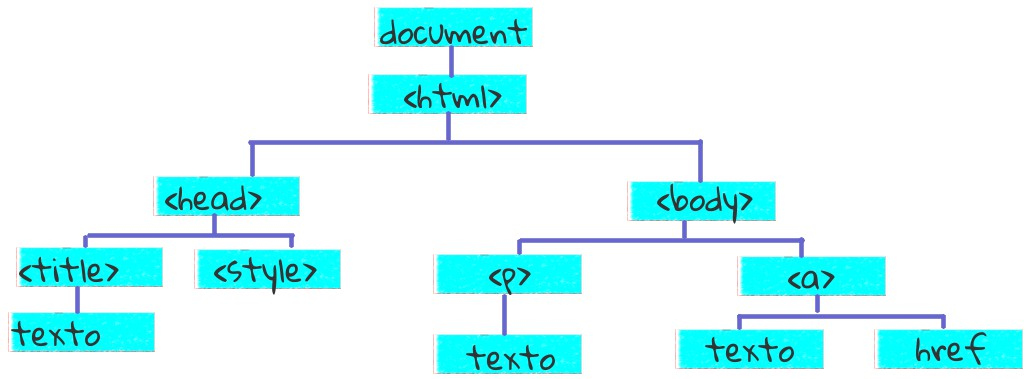
**Nodo padre:** el nodo padre es el nodo superior de un determinado nodo. Por ejemplo el nodo padre del nodo html es el nodo document; el nodo padre del nodo “title” es el nodo head; y así sucesivamente.

Un nodo sólo puede tener un nodo padre. El único caso especial es el no cabeza del árbol, que no tiene nodo padre

**Nodo hijo:** al contrario del nodo padre, el nodo dijo es el nodo siguiente en la estructura del árbol. Un nodo puede tener varios hijos o no tenerlos, como es el caso de las hojas del árbol.

Por ejemplo; la etiqueta a tiene dos nodos hijos (texto y href); a su vez, la etiqueta href no tiene hijos por ser una hoja del árbol.

De alguna manera se puede considerar que “document” tiene el resto de las etiquetas como nodos hijos.



ETIQUETAS HTML Y SUS ATRIBUTOS

Todas las etiquetas HTML tienen propiedades generales y específicas. Las propiedades o atributos generales son atributos comunes a todas las etiquetas HTML (estilo, visibilidad, id, clase, etc) y las específicas varían dependiendo de la etiqueta y su finalidad (width, height, backgroundColor, etc).

**Modificación de algunas propiedades**

Javascript permite, a través del DOM, crear o acceder a un elemento HTML y modificar sus propiedades.

Por ejemplo:

Podemos acceder al body de la página y cambiar su color de fondo.

document.body.style.backgroundColor='yellow'

o cambiar las propiedades de su fuente

document.body.style.color='#f0f'

document.body.style.fontFamily='Courier'

o incluso otras propiedades

document.body.style.margin='40px'

document.body.style.border='1px solid #f00'

document.body.style.padding='20px'

Observe que luego de tomar el body de la página, accedemos a su atributo style y específicamente a los elementos que este posee para asignarles un nuevo valor.

Ejercicio:

Copie el siguiente código en su editor:

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Hola mundo</title>

<style></style>

</head>

<body>

<p>Página para mostrar el DOM</p>

<a href="index.htm">Home</a>

<div id="primerdiv" class="paneles">Lorem ipsum dolor sit amet consectetur adipisicing elit. Quasi, laborum earum quia molestias saepe fugiat expedita accusamus tempora molestiae facere, ab sed repellat eos alias sequi similique distinctio porro at.</div>

<div id="segundodiv" class="paneles">

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Aspernatur fugit officia commodi expedita dicta voluptates consectetur reprehenderit quod. Optio minima molestias aliquid officia sunt voluptatibus nihil rem, et dolorum maxime?

</div>

<ol id="listaOrdenada"></ol>

<script>

//agregue el codigo en este lugar

</script>

</body>

</html>

Teniendo en cuenta la definición dada anteriormente del DOM y los métodos para acceder a los elementos del mismo. realicemos lo siguiente:

* Agregue la siguiente instrucción, ejecute y describa lo que sucede.

*console.log(document.body)*

* Agregue la siguiente instrucción, ejecute y describa lo que sucede.

*console.log(document.getElementbyId(‘primerdiv’)*

* **Propiedades de navegación entre los nodos de tipo elemento**

Cualquier nodo de tipo elemento dispone de las siguientes propiedades para obtener la referencia de:

* firstElementChild : almacena la referencia del primer nodo de tipo elemento hijo.
* parentElement : almacena la referencia del nodo elemento padre.
* lastElementChild : almacena la referencia del último nodo hijo.
* nextElementSibling : almacena la referencia del siguiente nodo de tipo elemento (si lo tiene)
* previousElementSibling : almacena la referencia del nodo de tipo elemento previo (si lo tiene)
* children : almacena una colección con todos los elementos tipo nodo hijo.

## **firstElementChild**

Como dijimos almacena la referencia del primer nodo hijo, Luego a partir del objeto 'document' accedemos a la propiedad 'body' y este al tratarse de un nodo de tipo elemento podemos acceder a la propiedad firstElementChild y obtener la referencia del primer div (el elemento body tiene 3 hijos, podemos contraer los nodos del árbol y verificar dicha situación), luego cambiamos el color de todo el contenido del div:

**const** div1=document.body.firstElementChild

div1.style.backgroundColor='red'

Ahora como el primer div es un nodo de tipo elemento y tiene 3 hijos de tipo nodo elemento (controlar el árbol que puede ver en la parte inferior a la derecha) podemos descender al último nodo mediante la propiedad lastElementChild

**const** parrafo3=div1.lastElementChild

parrafo3.style.backgroundColor='yellow'

Ahora obtenemos la referencia del nodo hermano siguiente al div1 mediante la propiedad 'nextElementSibling' (sibling=hermano) y procedemos a cambiar el color del elemento 'ol':

**const** ol=div1.nextElementSibling

ol.style.backgroundColor='blue'

Obtenemos la referencia del último elemento de tipo 'li' mediante la propiedad 'lastElementChild' y seguidamente retrocedemos al hermano anterior para cambiar el color del Item 3:

**const** li4=ol.lastElementChild

**const** li3=li4.previousElementSibling

li3.style.backgroundColor='green'

Ahora obtenemos la referencia del siguiente nodo al objeto 'ol', que se trata del segundo div. Luego de obtener la referencia del div procedemos a acceder a la propiedad children que almacena una colección con todos sus nodos de tipo elemento hijo. Podemos recorrer dicha colección mediante un for of:

**const** div2 = ol.nextElementSibling

**const** hijos = div2.children

**for** (**let** elemento of hijos) {

elemento.style.backgroundColor = 'yellow'

}

Por último vamos a obtener la referencia del nodo padre del div1 y cambiar el color (el padre del div1 es realmente el nodo elemento body):

**const** padre=div1.parentElement

padre.style.backgroundColor='black'

[TutorialesProgramacionYa.com](https://www.tutorialesprogramacionya.com/)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Prueba</title>

</head>

<body>

<div>

<p>Primer párrafo</p>

<p>Segundo párrafo</p>

<p>Tercer párrafo</p>

</div>

<ol>

<li>Item 1</li>

<li>Item 2</li>

<li>Item 3</li>

<li>Item 4</li>

</ol>

<div>

<a href="#">Enlace 1</a>

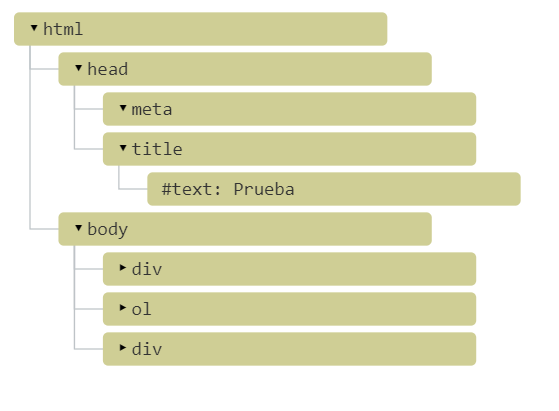
<a href="#">Enlace 2</a>

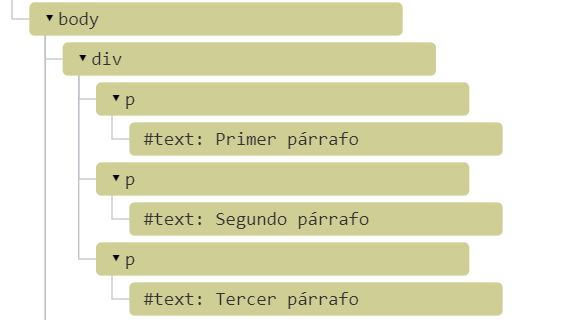
<a href="#">Enlace 3</a>

</div>

</body>

</html>





# **Buscar un elemento particular con el método getElementById**

Las propiedades de navegación que vimos en el concepto anterior se emplean cuando debemos acceder a un nodo de tipo elemento que se encuentra próximo a donde nos encontramos: un hijo, un padre o un hermano, pero que pasa si el elemento se encuentra lejos del elemento 'body', hay un método para posicionarnos en cualquier nodo del árbol.

El método en cuestión se llama 'getElementById' y requiere obligatoriamente que el elemento HTML cuente con la propiedad 'id'. El método getElementById pertenece al objeto 'document'.

Vamos a posicionarnos en el segundo item de la lista ordenada (ol), para ello hemos definido la propiedad id con el valor 'item2', ahora implementamos el siguiente código para acceder:

**const** li2 = document.**getElementById**('item2')

li2.style.backgroundColor = 'red'

Nos hemos posicionado directamente en el elemento de tipo li que tiene definida la propiedad id con el valor 'item2', luego todos los elementos 'id' de una página deben tener un valor distinto.

Ahora vamos a acceder a la celda de la tabla que tiene definida la propiedad id:

**const** celda = document.**getElementById**('celda')

celda.style.backgroundColor = 'yellow'

Una vez posicionado podemos utilizar las propiedades de navegación que ya conocemos:

**const** lista1=document.**getElementById**('lista1')

**for**(**let** item **of** lista1.children) {

item.style.fontFamily='Courier'

item.style.fontSize='3rem'

}

# **Nodos tipo elemento: propiedad innerHTML**

La propiedad innerHTML solo esta presente en los nodos de tipo elemento y no en los nodos de tipo texto. Si accedemos a su valor nos retorna un string con todos los nodos de tipo texto y nodos elementos que contiene hacia abajo en el árbol de nodos de la página web.

Por ejemplo si accedemos a la propiedad innerHTML del nodo elemento 'body' luego recuperamos un string con todo el contenido del body:

alert(document.body.innerHTML)

Como vemos nos aparece todo el contenido comprendido entre las etiquetas <body> y </body> sin incluirlas, luego de probar la salida de la función alert, puede borrarla para evitar que se abra dicho diálogo cada vez que se modifique el programa.

Pero no solo podemos consultar el valor almacenado en la propiedad innerHTML, sino que podemos asignarle un string con un bloque de HTML y el navegador se encarga de modificar el árbol de nodos y recrearlo con la nueva estructura.

Veamos como podemos cambiar el contenido de una lista desordenada (ul) mediante la asignación de un string que contienen las nuevas etiquetas a mostrar dentro de la lista:

setTimeout(() => {

const lista1 = document.querySelector("#lista1")

lista1.innerHTML = `

<**li**>one</**li**>

<**li**>two</**li**>

<**li**>three</**li**>

<**li**>four</**li**>`

}, 3000)

Vea luego de 3 segundo como la página web (la ventana que se encuentra abaja) se modifica su contenido (no he cambiado la página HTML original de la derecha, tener en cuenta que la propiedad innerHTML modifica el DOM en memoria y no modifica el archivo HTML físicamente)

Veamos otro ejemplo de como crear una tabla HTML en forma dinámica:

**let** tabla1 = '<table id="tabla1">'

**for**(**let** fila=1;fila<10;fila++) {

tabla1 += '<tr>'

**for**(**let** columna =1;columna<10;columna++) {

tabla1 += `<td>(${fila}-${columna})</td>`

}

tabla1 += '</tr>'

}

tabla1 += '</table>'

document.**querySelector**("#divtabla1").innerHTML=tabla1

document.**querySelector**("#tabla1").style.border='1px solid #f00'

document.**querySelector**("#tabla1").style.borderCollapse='collapse'

Podemos generar un string que contenga etiquetas HTML que luego son creados los nodos en el momento que le asignamos a la propiedad innerHTML.

Veamos otro ejemplo de actualizar un div con la hora actual de la computadora:

setInterval(() => {

**const** fechaActual = **new** **Date**()

document.**querySelector**("#hora").innerHTML = `${fechaActual.getHours()}:${fechaActual.getMinutes()}:${fechaActual.getSeconds()}`

}, 1000)

# **Nodos tipo texto: propiedad nodeValue**

Vimos que las propiedades innerHTML y outerHTML son solo válidas para los nodo tipo elemento, ahora veremos que para los nodos de tipo texto podemos acceder a su contenido o actualizar su contenido mediante la propiedad 'nodeValue'

Primero debemos descender de un nodo de tipo elemento al nodo de texto mediante la propiedad firstChild y no firstElementChild (tener en cuenta que firstChild desciende al primer nodo de tipo texto o elemento y como tenemos solo un nodo de texto es nuestra solución), luego que tenemos la referencia del nodo de tipo texto podemos acceder al texto mediante nodeValue.

Veamos como intercambiamos los textos de dos párrafos accediendo a la propiedad nodeValue. Obtenemos la referencia del node de tipo elemento mediante querySelector y accedemos a la propiedad firstChild de dicho nodo, el cual tiene la referencia del nodo hijo:

**const** parrafo1 = document.**querySelector**("#parrafo1").firstChild

**const** parrafo2 = document.**querySelector**("#parrafo2").firstChild

**const** texto = parrafo1.nodeValue

parrafo1.nodeValue = parrafo2.nodeValue

parrafo2.nodeValue = texto

Todos los nodos disponen de una propiedad llamada 'nodeName', en el caso de tratarse de un nodo de texto, este almacena el valor '#text' y en el caso de tratarse de un nodo de tipo elemento, la propiedad almacena el nombre de la etiqueta HTML en mayúsculas:

parrafo1.nodeValue = 'Nodo de texto: ' + parrafo1.nodeName

parrafo2.nodeValue = 'Nodo elemento: ' + document.body.firstElementChild.nodeName

# **Propiedad textContent**

La propiedad 'textContent' la tienen tanto los nodos de tipo elemento como los nodos de tipo texto.

'textContent' nos permite acceder al contenido de texto del nodo que hacemos referencia en el caso que se trate de un nodo valor y si se trata de un nodo elemento hace referencia al texto de sus descendientes.

Podemos extraer de un nodo de tipo elemento solo los textos de todos sus descendientes obviando las etiquetas HTML:

**const** resultado=document.**querySelector**("#resultado")

resultado.textContent=document.**querySelector**("#lista1").textContent

De la lista1:

<ol id="lista1">

<li>papas</li>

<li>manzanas</li>

<li id="item3">peras</li>

</ol>

Solo extrae el texto de cada uno de los elementos hijos, y luego asignamos a la propiedad textContent del elemento div:

papas manzanas peras

Hay una gran diferencia de extraer el contenido de texto con la propiedad 'textContent' que con la propiedad 'innerHTML':

resultado.textContent=document.querySelector("#lista1").innerHTML

Como vemos la propiedad innerHTML retorna todo el texto y las etiquetas HTML que envuelven dichos textos:

<**li**>papas</**li**>

<**li**>manzanas</**li**>

<**li** id="item3">peras</**li**>

Ahora podemos ver también como cuando asignamos a la propiedad textContent un string que contiene etiquetas HTML luego las mismas no modifican la estructura del árbol sino que las tratan como texto.

Si en cambio cargamos el string devuelto por la propiedad innerHTML a la propiedad innerHTML, tenemos como resultado que se modifica el DOM, agregando todas las marcas:

resultado.innerHTML=document.querySelector("#lista1").innerHTML

La propiedad 'innerHTML' procesa y crea los nodos indicados en el string, en cambio si cambiamos la propiedad 'textContent', si le asignamos un string que contienen etiquetas, las mismas son tratadas como texto.

Siempre que necesitemos cargar solo texto en un nodo, debemos asignar a la propiedad textContent y evitar de asignarlo a innerHTML. Ya que el texto no es procesado y es más probable que tenga mejor rendimiento (no se crean otros nodos tipo elemento). Además, esto evita ataques [XSS](https://es.wikipedia.org/wiki/Cross-site_scripting).

Es decir que modificar el textContent permite escribir texto de "forma seguro".

Podemos cambiar el texto de un nodo de texto, accediendo a la propiedad textContent:

**const** item1=document.**querySelector**("#item3").firstChild.textContent = "Hola mundo"