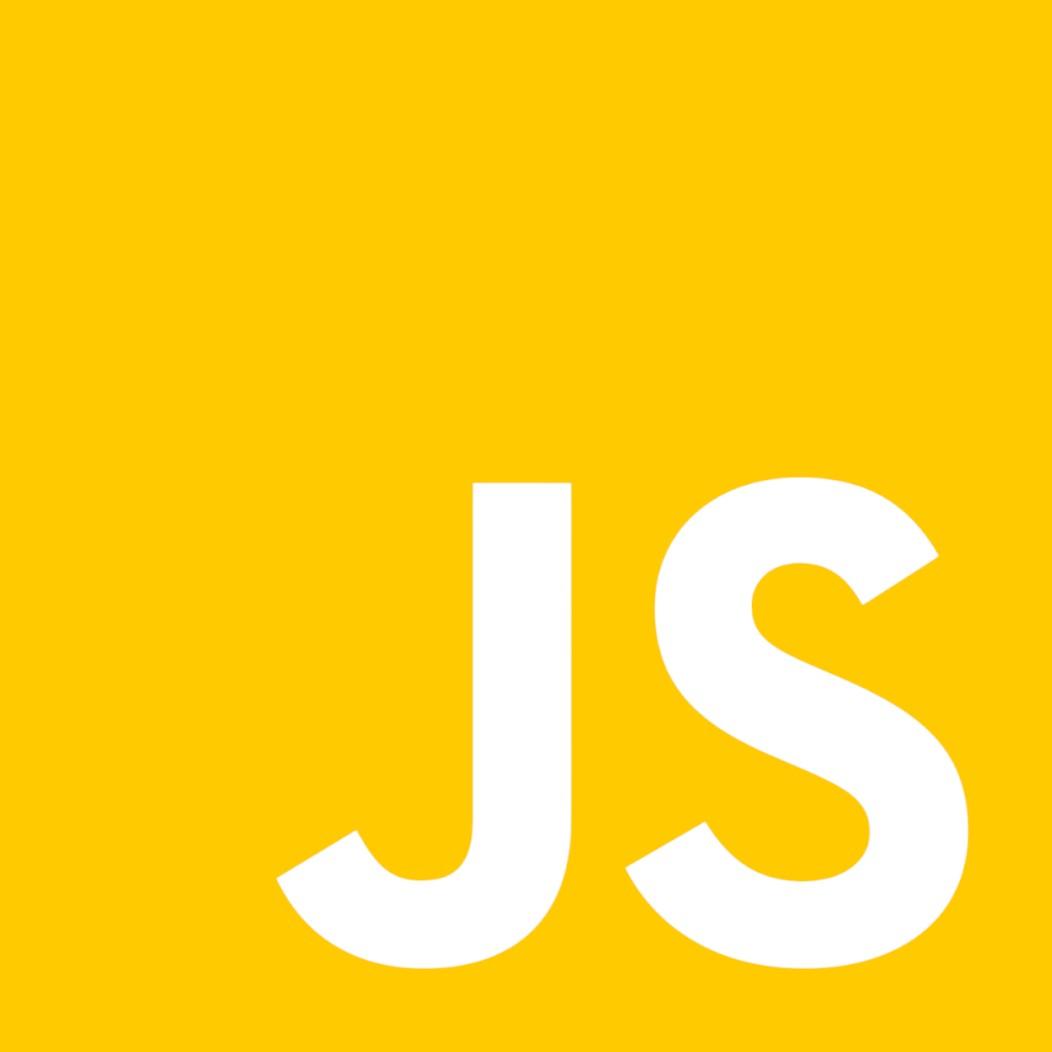
💻 JAVASCRIPT

****

**JAVASCRIPT**

**Apuntes y Documentación**

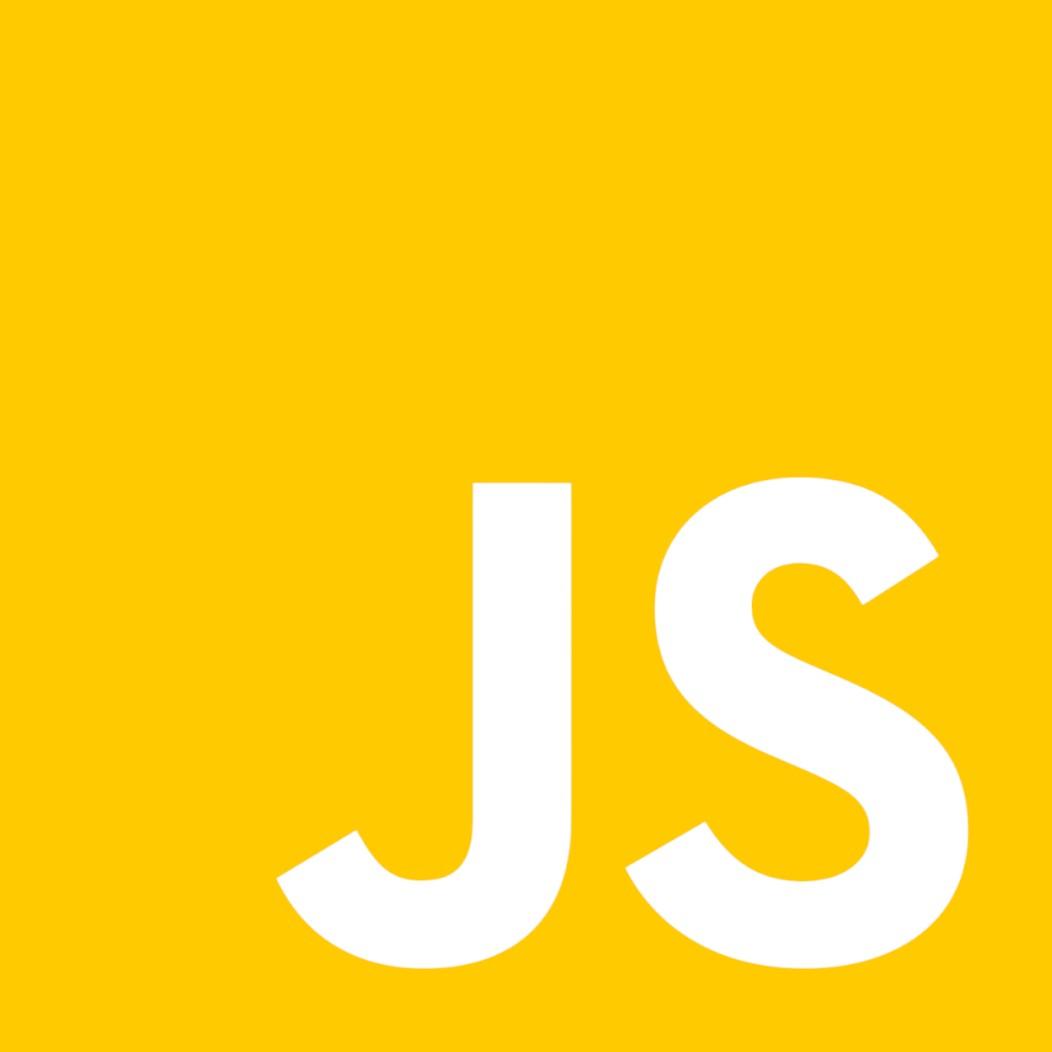
**Realizados por:** Lucas Falla Urtiaga.

[**https://es.javascript.info/browser-environment#dom-modelo-de-objetos-del-documento**](https://es.javascript.info/browser-environment#dom-modelo-de-objetos-del-documento)

[**https://developer.mozilla.org/es/docs/Web/API/Document/querySelector**](https://developer.mozilla.org/es/docs/Web/API/Document/querySelector)

**Página usada para el logo:** Photoroom

1️⃣ Documentos

****

**APARTADO 1: DOCUMENTOS**

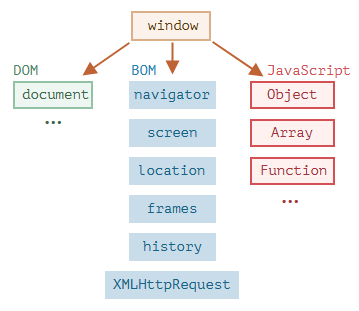
# **1 ENTORNO DEL NAVEGADOR Y ESPECIFICACIONES**

El lenguaje JavaScript fue creado inicialmente para los navegadores web. Desde entonces, ha evolucionado en un lenguaje con muchos usos y plataformas.

Una plataforma puede ser un navegador, un servidor web u otro host (*“anfitrión”*); incluso una máquina de café *“inteligente”*, puede ejecutar JavaScript. Cada uno de ellos proporciona una funcionalidad específica de la plataforma. La especificación de JavaScript llama a esto entorno de host.

Un entorno host proporciona sus propios objetos y funciones adicionales al núcleo del lenguaje. Los navegadores web proporcionan un medio para controlar las páginas web. Node.js proporciona características del lado del servidor junto a muchas otras.

**Esta es una vista general de lo que tenemos cuando JavaScript se ejecuta en un navegador web:**



## ***1.1- Window***

Hay un objeto “raíz” llamado **window** que tiene dos roles:

* Es un objeto global para el código JavaScript, como se describe en el capítulo Objeto Global.
* Representa la *“ventana del navegador”* y proporciona métodos para controlarla.

Ejemplo de uso como objeto global:

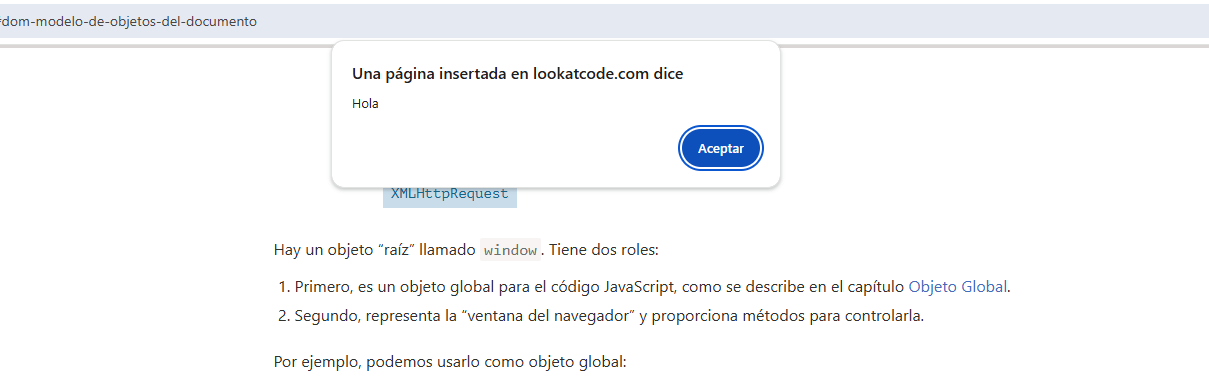
function sayHi() {

alert("Hola");

}

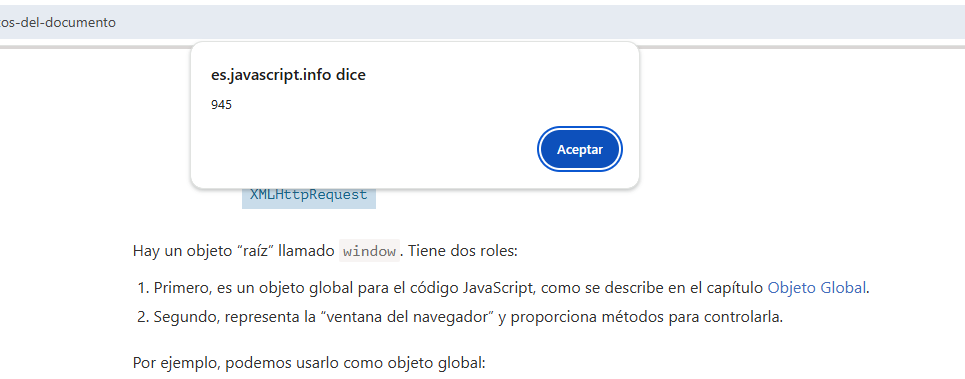
// Las funciones globales son métodos del objeto global:

window.sayHi();



Ejemplo de uso para saber (entre otros) la altura una ventana del navegador:

alert(window.innerHeight); // Muestra la altura interior de la ventana.

****

Hay más métodos y propiedades específicos de **window** que se cubrirán más adelante.

## ***1.2- DOM (Modelo de Objetos del Documento)***

**Document Object Model**, o **DOM**, representa todo el contenido de la página como objetos que pueden ser modificados.

El objeto **document** es el punto de entrada a la página, y con él podemos cambiar o crear cualquier cosa en esta.

Por ejemplo:

// Cambia el color de fondo de la página a rojo.

document.body.style.background = "red";

// Esta línea deshace el cambio después de 1 segundo.

setTimeout(() => document.body.style.background = "", 1000);



*(Tras un segundo volvería a la normalidad).*

Aquí se ha usado *“document.body.style”*, pero se pueden acceder a muchas más propiedades del elemento *body*.

## ***1.3- BOM (Modelo de Objetos del Navegador)***

El **Modelo de Objetos del Navegador (Browser Object Model, BOM)** son objetos adicionales proporcionados por el navegador (entorno host) para trabajar con todo aquello que no esté relacionado con el documento.

Por ejemplo:

El objeto **navigator** proporciona información sobre el navegador y el sistema operativo. Hay muchas propiedades, pero las dos más conocidas son:

* **navigator.userAgent:** acerca del navegador actual.
* **navigator.platform:** acerca de la plataforma (ayuda a distinguir Windows/Linux/Mac...).

El objeto **location** nos permite leer la URL actual y puede redirigir el navegador a una nueva.

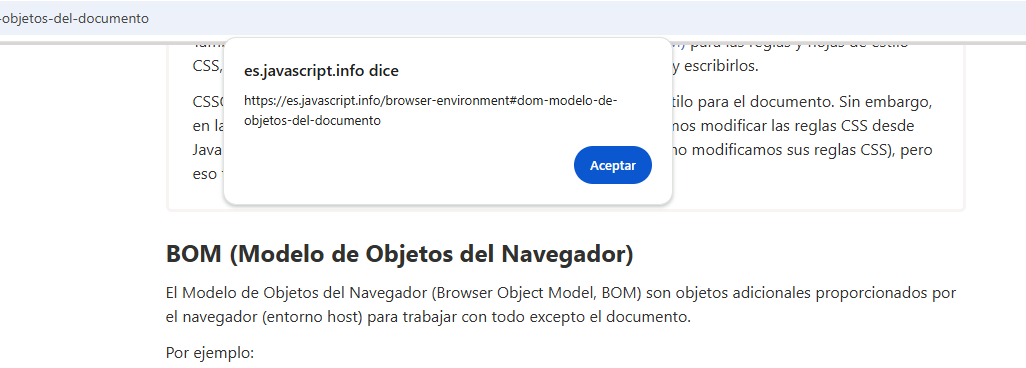
Aquí vemos cómo podemos usar el objeto *location*:

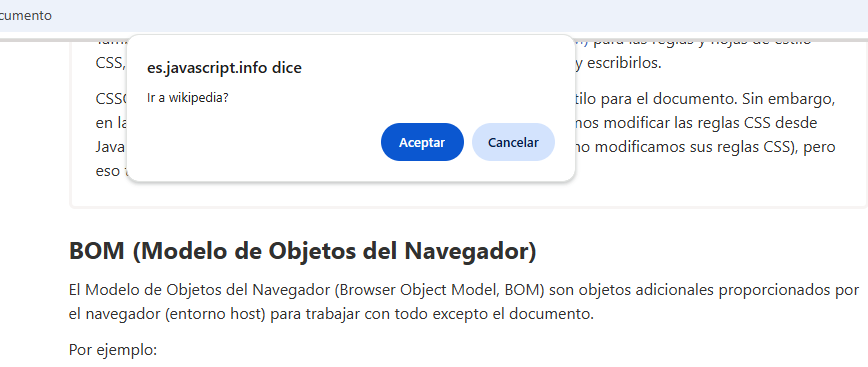
alert(location.href); // Muestra la URL actual.

if (confirm("Ir a wikipedia?")) {

location.href = "https://wikipedia.org"; // Redirigir el navegador a la URL de Wikipedia.

}





*(Si le diéramos a “Aceptar” nos llevaría a la página oficial de Wikipedia, si no, no haría nada).*

Las funciones **alert/confirm/prompt** también forman parte de BOM: no están directamente relacionadas con el documento, sino que representan métodos puros de comunicación del navegador con el usuario.

# **2 ÁRBOL DEL MODELO DE OBJETOS DEL DOCUMENTO (DOM)**

La estructura de un documento HTML son las etiquetas.

**Según el DOM, cada etiqueta HTML es un objeto. Las etiquetas anidadas son llamadas *“hijas”* de la etiqueta que las contiene. El texto dentro de una etiqueta también es un objeto.**

Todos estos objetos son accesibles empleando JavaScript, y podemos usarlos para modificar la página.

Por ejemplo, *document.body* es el objeto que representa la etiqueta <body>.

// Cambia el color de fondo a rojo.

document.body.style.background = "red";

// Deshace el cambio después de 3 segundos.

//setTimeout(() => document.body.style.background = "", 3000);

En el caso anterior usamos *“style.background”* para cambiar el color de fondo del *document.body*, pero existen otras propiedades, tales como:

* **innerHTML**: contenido HTML del nodo.
* **offsetWidth**: ancho del nodo (en píxeles).
* Muchas más XD.

Más adelante, aprenderemos otras formas de manipular el DOM, pero primero necesitamos conocer su estructura.

## ***2.1- Estructura del DOM***

<!DOCTYPE HTML>

<html>

<head>

<title>About elk</title>

</head>

<body>

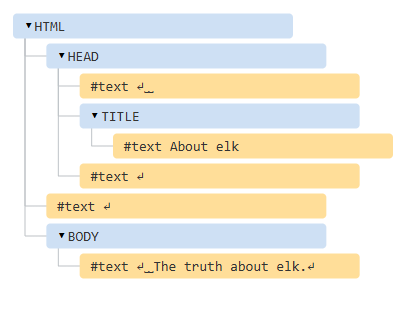
The truth about elk.

</body>

</html>

**El DOM representa el HTML como una estructura de árbol de etiquetas.**

A continuación podemos ver cómo se muestra:



**IMPORTANTE:** Cada nodo del árbol es un objeto.

Las etiquetas son nodos de elementos (o simplemente *“elementos”*) y forman la estructura del árbol. <html> está ubicado en la raíz del documento, por lo tanto, <head> y <body> son sus hijos y así sucesivamente.

El texto dentro de los elementos forma nodos de texto, y son etiquetados como *#text*. Un nodo de texto puede contener únicamente una cadena y no puede tener hijos, siempre es una hoja del árbol.

// Por ejemplo, la etiqueta <title> es un elemento que contiene tiene el texto *"About elk"*, que es un nodo de texto.

Además, también hay que tener en cuenta dos caracteres especiales en los nodos de texto:

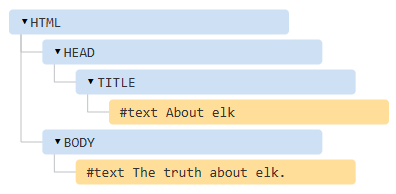
* **Un salto de línea: ↵** (en JavaScript se emplea “\n” para ello).
* **Un espacio: ␣**

Los espacios y líneas nuevas son caracteres totalmente válidos, al igual que letras y dígitos, por lo que ellos también forman nodos de texto y se convierten en parte del DOM. Así, por ejemplo, en el caso anterior la etiqueta <head> contiene un salto de línea y 2 espacios antes de la etiqueta <title>, por lo que estos pasan a convertirse en texto, apareciendo así un nodo *#text* que contiene esa nueva línea y dichos espacios.

En el siguiente ejemplo no hay nodos de texto con espacios en blanco:

<!DOCTYPE HTML>

<html><head><title>About elk</title></head><body>The truth about elk.</body></html>



// Como curiosidad, los espacios y saltos de línea ubicados antes de la etiqueta <head> son ignorados, y si colocamos algo después de la etiqueta </body>, automáticamente se situará dentro de body.

## ***2.2- Otros tipos de nodos***

Existen otros tipos de nodos además de elementos y nodos de texto. Por ejemplo, los **comentarios**:

<!DOCTYPE HTML>

<html>

<body>

The truth about elk.

<ol>

<li>An elk is a smart</li>

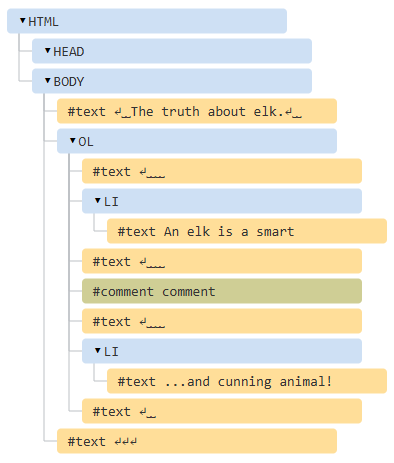
<!-- comment -->

<li>...and cunning animal!</li>

</ol>

</body>

</html>



Aquí se puede visualizar un nodo de comentario, etiquetado como *#comment*, entre dos nodos de texto.

Aunque los comentarios no afectan a la representación de la página de ninguna manera, existe la siguiente regla:

SI ALGO SE ENCUENTRA EN EL CÓDIGO HTML, ENTONCES TAMBIÉN DEBE ENCONTRARSE EN EL ÁRBOL DOM.

En total existen 12 tipos de nodos, pero en la práctica generalmente se trabaja con 4 de ellos:

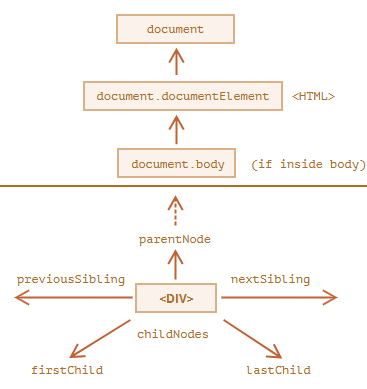
* **document**: el “punto de entrada” en el DOM.
* **nodos de elementos:** etiquetas HTML.
* **nodos de texto**: contienen texto.
* **comentarios:** permitan colocar información en el punto en el que se encuentren. No se mostrará, pero JS puede leerla desde el DOM.

# **3 RECORRIENDO EL DOM**

El **DOM** nos permite hacer cualquier cosa con sus elementos y contenidos, pero lo básico que debemos saber es cómo llegar al objeto que queremos manipular.

Todas las operaciones en el DOM comienzan con el objeto ***document****.* Este es el principal *“punto de entrada”* y con el que se puede acceder a cualquier nodo.

Esta imagen representa los enlaces que nos permiten viajar a través de los nodos del DOM:



## ***3.1- En la parte superior: documentElement y body***

Los tres nodos superiores están disponibles como propiedades de *document*:

* **<html>** = document.documentElement.
* **<body>** = document.body.
* **<head>** = document.head.

**// ¡CUIDADO! document.body puede ser null**

Un script no puede acceder a un elemento que no existe en el momento de su ejecución. Por ejemplo, si un script está dentro de <head>, entonces document.body no está disponible, porque el navegador no lo ha leído aún y devolvería ***null***.

// En el DOM, el valor *null* significa que *“no existe”* o *“no hay tal nodo”*.

## ***3.2- Hijos: childNodes, firstChild, lastChild***

Existen dos términos de vital importancia si se habla de hijos:

* **Nodos hijos (childNodes):** se trata de elementos que son hijos directos, es decir, sus descendientes inmediatos. Por ejemplo, <head> y <body> son hijos del elemento <html>.
* **Descendientes:** son todos los elementos anidados de un elemento dado, incluyendo los hijos, los hijos de sus hijos y así sucesivamente.

Por ejemplo, aquí <body> tiene de hijos <div> y <ul> (y unos pocos nodos de texto en blanco):

<html>

<body>

<div>Begin</div>

<ul>

<li>

<b>Information</b>

</li>

</ul>

</body>

</html>

Y los descendientes de <body> no son solo los hijos <div> y <ul> sino también elementos anidados más profundamente, como <li> (un hijo de <ul>) o <b> (un hijo de <li>) – el subárbol entero.

La colección childNodes enumera todos los nodos hijos, incluidos los nodos de texto.

El ejemplo inferior muestra todos los hijos de *document.body*:

<html>

<body>

<div>Begin</div>

<ul>

<li>Information</li>

</ul>

<div>End</div>

<script>

for (let i = 0; i < document.body.childNodes.length; i++) {

alert( document.body.childNodes[i] );

/\* Devuelve:

[object Text], [object HTMLDivElement], [object Text], [object HTMLUListElement],

[object Text], [object HTMLDivElement], [object Text], [object HTMLScriptElement] \*/

}

</script>

Más cosas...

</body>

</html>

Si se ejecuta el ejemplo anterior, el último elemento que se muestra es <script>. De hecho, el documento tiene más cosas debajo, pero en el momento de ejecución el navegador todavía no lo ha leído, por lo que el script no lo ve.

Las propiedades firstChild y lastChild dan acceso rápido al primer y al último hijo, ACTUANDO COMO ATAJOS.

También hay una función especial **elem.hasChildNodes()** para comprobar si existen nodos hijos.

## ***3.3- Colecciones del DOM***

Como podemos ver, **childNodes** parece un array. Pero realmente no es un array, sino más bien una colección: un objeto especial iterable.

Hay dos importantes consecuencias acerca de esto:

* Podemos usar *for..of* para iterar sobre él.

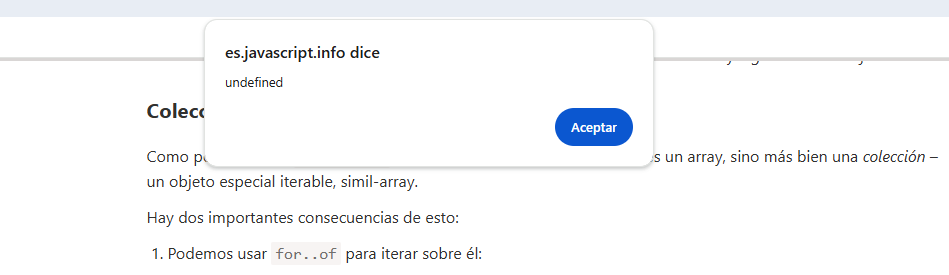
for (let elem of document.body.children) {

alert(elem); // Enseña todos los nodos de la colección.

}

* Los métodos de Array no funcionan, porque NO es un array (si intentamos utilizar sobre childNodes el método de arrays *“filter”*, este va a devolver *undefined*, porque repito, childNodes **NO ES UN ARRAY**.

alert(document.body.childNodes.filter); // undefined (¡No hay método filter!)



Aunque sí que podemos usar el método **Array.from()** para crear un array *“auténtico”* desde la colección si es que queremos usar métodos del array:

alert( Array.from(document.body.childNodes).filter ); // Función.

**// ¡CUIDADO! Las colecciones DOM son solo de lectura**

Las colecciones DOM, incluso más, **TODAS** las propiedades de navegación enumeradas en este capítulo son sólo de lectura.

No podemos reemplazar a un hijo por otro elemento asignándolo así childNodes[i] = ....

Cambiar el DOM necesita otros métodos.

**// ¡CUIDADO! Las colecciones del DOM están vivas**

Casi todas las colecciones del DOM, salvo algunas excepciones, están vivas. En otras palabras, reflejan el estado actual del DOM.

Si mantenemos una referencia a *elem.childNodes*, y añadimos o quitamos nodos del DOM, entonces estos nodos aparecen en la colección automáticamente.

**// ¡CUIDADO! No uses for..in para recorrer colecciones**

Las colecciones son iterables usando *for..of*, y aunque algunas veces alguien trata de utilizar *for..in* para eso, no es que sea especialmente correcto. El bucle *for..in* itera sobre todas las propiedades enumerables, y las colecciones tienen unas propiedades *“extra”* raramente usadas que normalmente no queremos obtener. Literalmente contienen palabras como *“entries”*, *“keys”*, *“values”*, *“length”* o *“item”*.

<body>

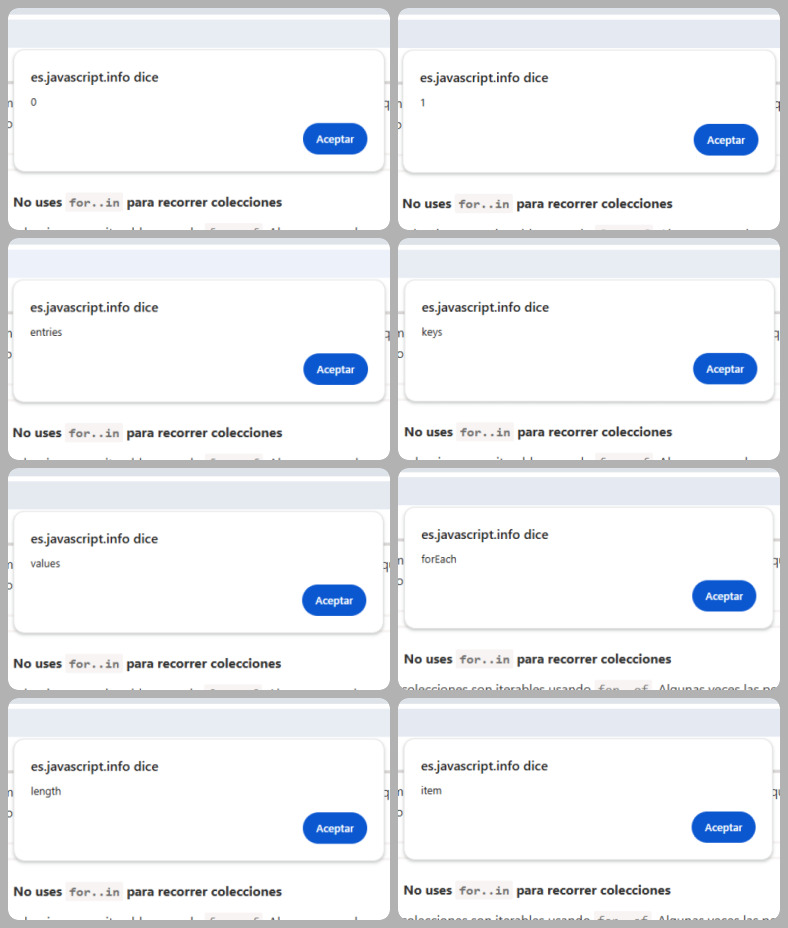
<script>

// Con for..in se enseñan: 0, 1, longitud, item, valores y multitud de otras cosas.

for (let prop in document.body.childNodes) alert(prop);

</script>

</body>



## ***3.4- Hermanos y el padre***

**Los hermanos son nodos que son hijos del mismo padre.**

Por ejemplo, normalmente <head> y <body> son hermanos.

<html>

<head>...</head><body>...</body>

</html>

* <body> se dice que es el hermano *“siguiente”* o a la *“derecha”* de <head>.
* <head> se dice que es el hermano *“anterior”* o a la *“izquierda”* de <body>.

El hermano **siguiente** está en la propiedad **nextSibling** y el **anterior** en **previousSibling**.

El **padre** está disponible en **parentNode**.

// El padre de <body> es <html>

alert( document.body.parentNode === document.documentElement ); // verdadero

// Después de <head> va <body>

alert( document.head.nextSibling ); // HTMLBodyElement

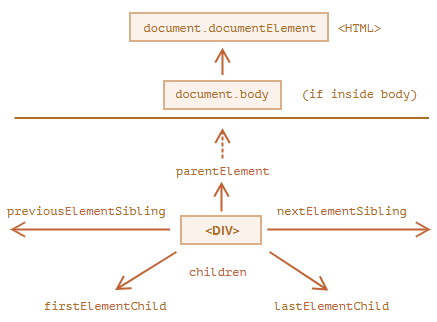
// Antes de <body> va <head>

alert( document.body.previousSibling ); // HTMLHeadElement

## ***3.5- Navegación sólo por elementos***

Las propiedades de navegación enumeradas abajo se refieren a todos los nodos. Por ejemplo, en childNodes podemos ver nodos de texto, nodos elementos, y si existen, incluso los nodos de comentarios.

**Pero estos enlaces de navegación SÓLO tienen en cuenta los nodos elementos:**

****

Los enlaces son similares a los de arriba, solo que tienen dentro la palabra **Element**:

* **children:** son esos hijos que tienen el elemento nodo.
* **firstElementChild, lastElementChild:** el primer y el último elemento hijo.
* **previousElementSibling, nextElementSibling:** elementos vecinos.
* **parentElement:** elemento padre.

**// ¡INFORMACIÓN ÚTIL! ¿Por qué parentElement? ¿Puede el padre no ser un elemento?**

La propiedad parentElement devuelve el *“elemento”* padre, mientras parentNode devuelve *“cualquier nodo”* padre. Estas propiedades son normalmente las mismas: ambas seleccionan el padre. Con la excepción de document.documentElement:

* **alert( document.documentElement.parentNode ); // documento**
* **alert( document.documentElement.parentElement ); // null**

La razón es que el nodo raíz document.documentElement (<html>) tiene a document como su padre. Pero document no es un elemento nodo, por lo que parentNode lo devuelve y parentElement no lo hace.

Una vez visto lo anterior, vamos a modificar uno de los ejemplos de arriba reemplazando **childNodes** por **children**:

**childNodes**

<html>

<body>

<div>Begin</div>

<ul>

<li>Information</li>

</ul>

<div>End</div>

<script>

for (let i = 0; i < document.body.childNodes.length; i++) {

alert( document.body.childNodes[i] );

/\* Devuelve:

[object Text], [object HTMLDivElement], [object Text], [object HTMLUListElement],

[object Text], [object HTMLDivElement], [object Text], [object HTMLScriptElement] \*/

}

</script>

Más cosas...

</body>

</html>

**children**

<html>

<body>

<div>Begin</div>

<ul>

<li>Information</li>

</ul>

<div>End</div>

<script>

for (let elem of document.body.children) {

alert(elem);

/\* Devuelve:

[object HTMLDivElement], [object HTMLUListElement], [object HTMLDivElement],

[object HTMLScriptElement] \*/

}

</script>

Más cosas...

</body>

</html>

Como se ha podido apreciar, mientras que childNodes ha mostrado todos los los nodos (tanto de texto como de elemento y demás) children sólo ha mostrado los nodos de elemento.

## ***3.6- Tablas***

Ciertos elementos del DOM pueden tener propiedades adicionales, específicas de su tipo, por conveniencia.

Las tablas son un ejemplo de ello, y representan un particular caso importante:

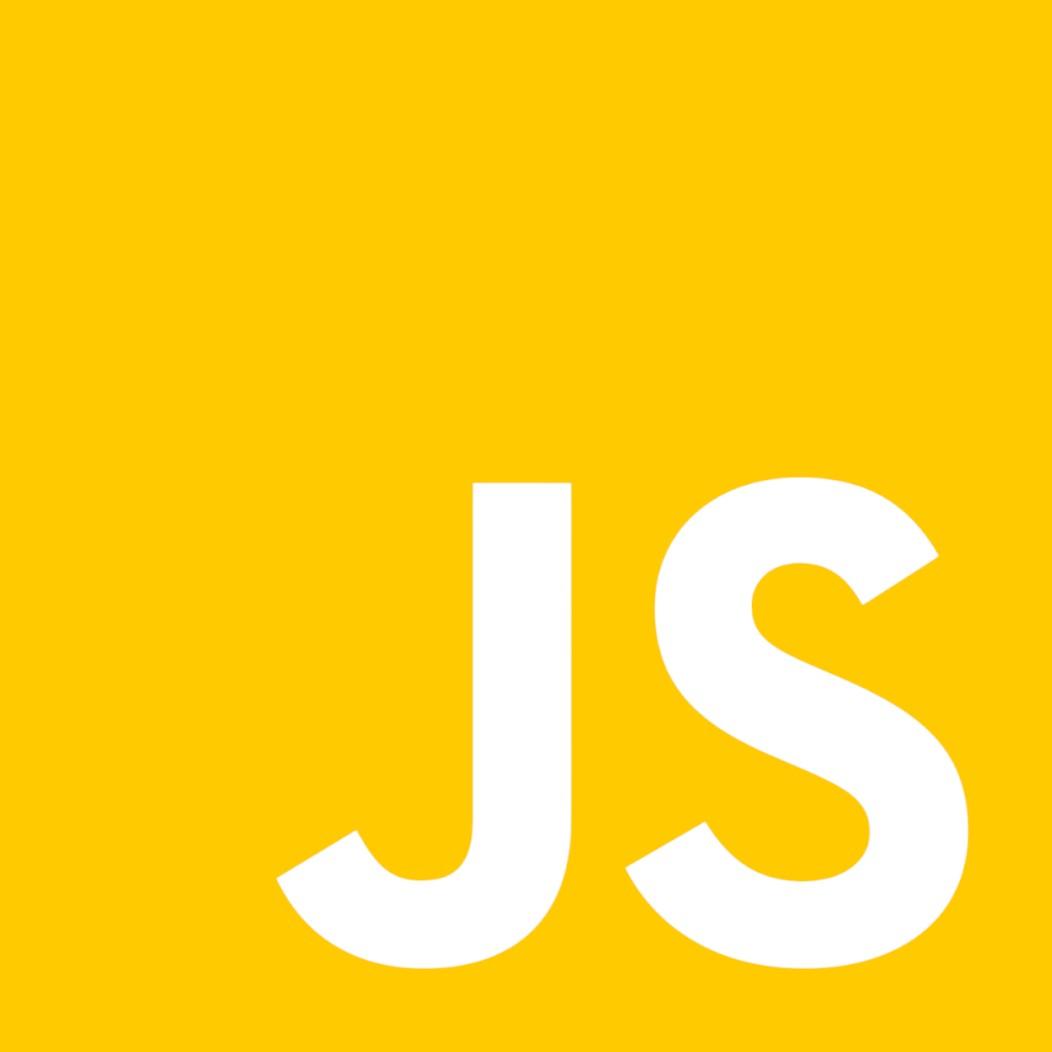
El elemento <table> soporta estas otras propiedades (además de las mencionadas anteriormente):

* **table.rows:** es la colección de elementos <tr> (filas) de la tabla.
* **table.caption / table.tHead / table.tFoot:** referencian a los elementos <caption>, <thead> y <tfoot> respectivamente.
* **table.tBodies**: se trata de la colección de elementos <tbody> (pueden ser muchos según el estándar, pero siempre habrá al menos uno, aunque no esté en el HTML el navegador lo pondrá en el DOM).

**TERMINAR TABLAS**

# 

➕ Métodos útiles

****

**APARTADO EXTRA: MÉTODOS ÚTILES**

# **1 ENTORNO DEL NAVEGADOR Y ESPECIFICACIONES**

**document.getElementById(‘id’):** devuelve el elemento de dicho id.

<div id="elem">

<div id="elem-content">Elemento</div>

</div>

// Obtiene el elemento.

let elem = document.getElementById('elem');

// Hace que su fondo sea rojo.

elem.style.background = 'red';

**elem.querySelector(‘css’):** devuelve el primer elemento que coincide con el selector CSS dado.

<div class="user-panel main">

<input name="login" style="background-color:red" />

<input name="login" style="background-color:blue" />

</div>

<script>

// Devuelve el primer elemento <input name="login" /> dentro de <div class="user-panel

main" (en nuestro caso, el de fondo rojo).

var el = document.querySelector("div.user-panel.main input[name='login']");

el.value = "TREMENDO."

</script>



**elem.querySelectorAll(‘css’):** devuelve todos los elementos (un array) dentro de *elem* que coinciden con el selector CSS dado.

<div class="user-panel main">

<input name="login" />

</div>

<script>

// Devuelve el primer elemento <input name="login" /> dentro de <div class="user-panel

main">.

var el = document.querySelector("div.user-panel.main input[name='login']");

</script>

**var\_dump(var):**devuelve el tipo de una variable.

**strlen(String):** devuelve la longitud de una cadena de caracteres.

**str\_word\_count(String):** cuenta el número de palabras que tiene el String.

**strpos(String):** busca un texto específico dentro de una cadena. Si se encuentra una coincidencia, la función devuelve la posición del primer carácter coincidente. Si no se encuentra ninguna, devolverá *False*.

**is\_int(int):** devuelve si el tipo de la variable es un entero.

(Los métodos **is\_integer(int)** y **is\_long(int)** hacen exactamente lo mismo).

# ***1- Modificar Cadenas***

**strtoupper(String):** devuelve el String en *UPPERCASE*.

**strtolower(String):** devuelve el String en *lowercase*.

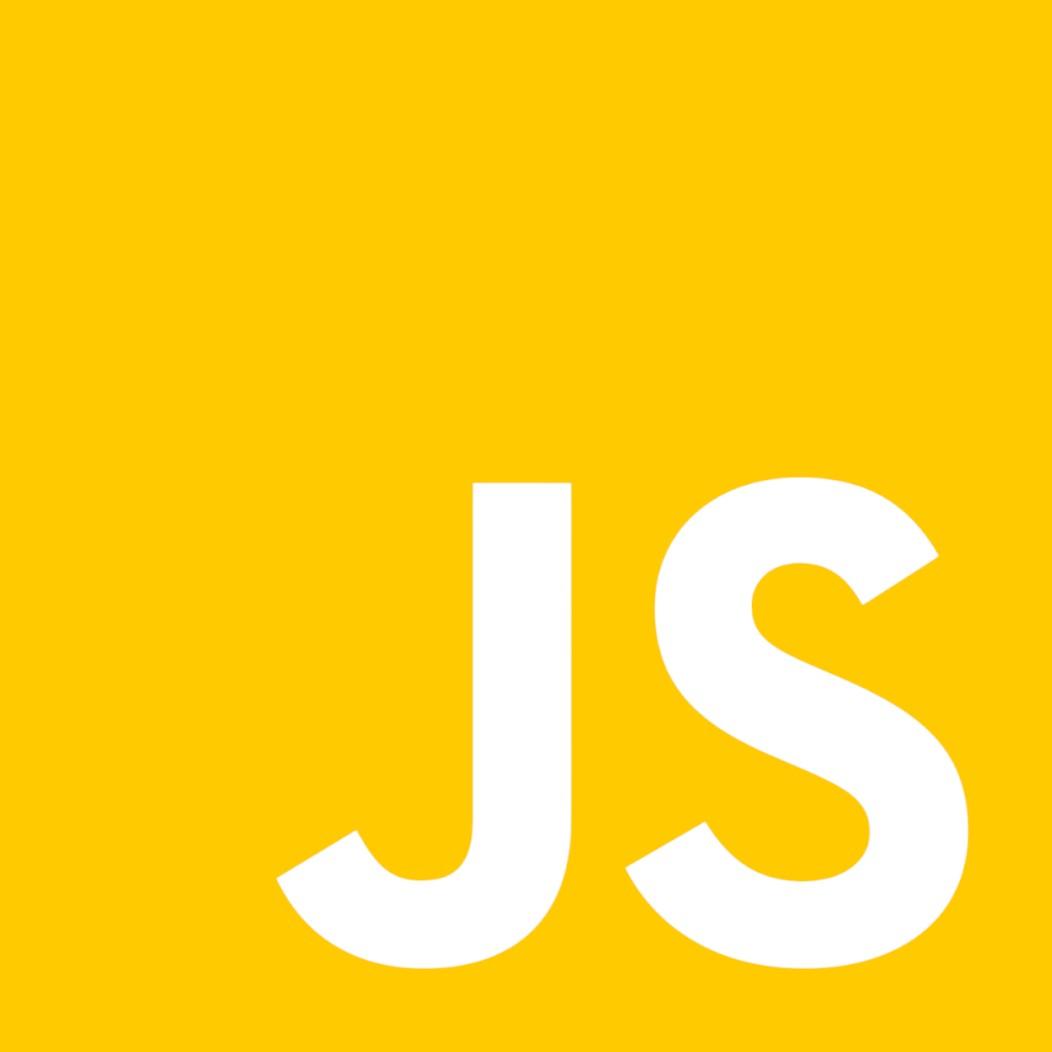
**str\_replace("World", "Dolly", “Hello World!”):** reemplaza los caracteres de una cadena por otros. En este caso devolvería: *“Hello Dolly!”*.

**strrev(String):** devuelve la cadena de caracteres invertida.

**trim(“ Hello World ”):** elimina cualquier espacio en blanco del principio o del final de la cadena. En este caso devolvería: *“Hello World”* en lugar de *“ Hello World ”*.

**explode(“ “, “Hello World!”):** divide la cadena en un array según el separador impuesto. En este caso devolvería: *Array( [“Hello”], [“World!”] )*.

➕ Extras

****

**APARTADO EXTRA: EXTRAS**

# **! AVISOS DE CUIDADO**

**// ¡CUIDADO! document.body puede ser null**

Un script no puede acceder a un elemento que no existe en el momento de su ejecución. Por ejemplo, si un script está dentro de <head>, entonces document.body no está disponible, porque el navegador no lo ha leído aún y devolvería ***null***.

En el DOM, el valor null significa que no existe tal nodo.

**// ¡CUIDADO! Las colecciones DOM son solo de lectura**

Las colecciones DOM, incluso más, **TODAS** las propiedades de navegación enumeradas en este capítulo son sólo de lectura.

No podemos reemplazar a un hijo por otro elemento asignándolo así childNodes[i] = ....

Cambiar el DOM necesita otros métodos.

**// ¡CUIDADO! Las colecciones del DOM están vivas**

Casi todas las colecciones del DOM, salvo algunas excepciones, están vivas. En otras palabras, reflejan el estado actual del DOM.

Si mantenemos una referencia a *elem.childNodes*, y añadimos o quitamos nodos del DOM, entonces estos nodos aparecen en la colección automáticamente.

**// ¡CUIDADO! No uses for..in para recorrer colecciones**

Las colecciones son iterables usando *for..of*. Algunas veces las personas tratan de utilizar *for..in* para eso. Por favor, no lo hagas. El bucle *for..in* itera sobre todas las propiedades enumerables. Y las colecciones tienen unas propiedades *“extra”* raramente usadas que normalmente no queremos obtener.

# **-> RESUMEN**

Las propiedades y métodos se describen en la especificación: DOM Living Standard.

**—---------------------------------------------------------------------------------**

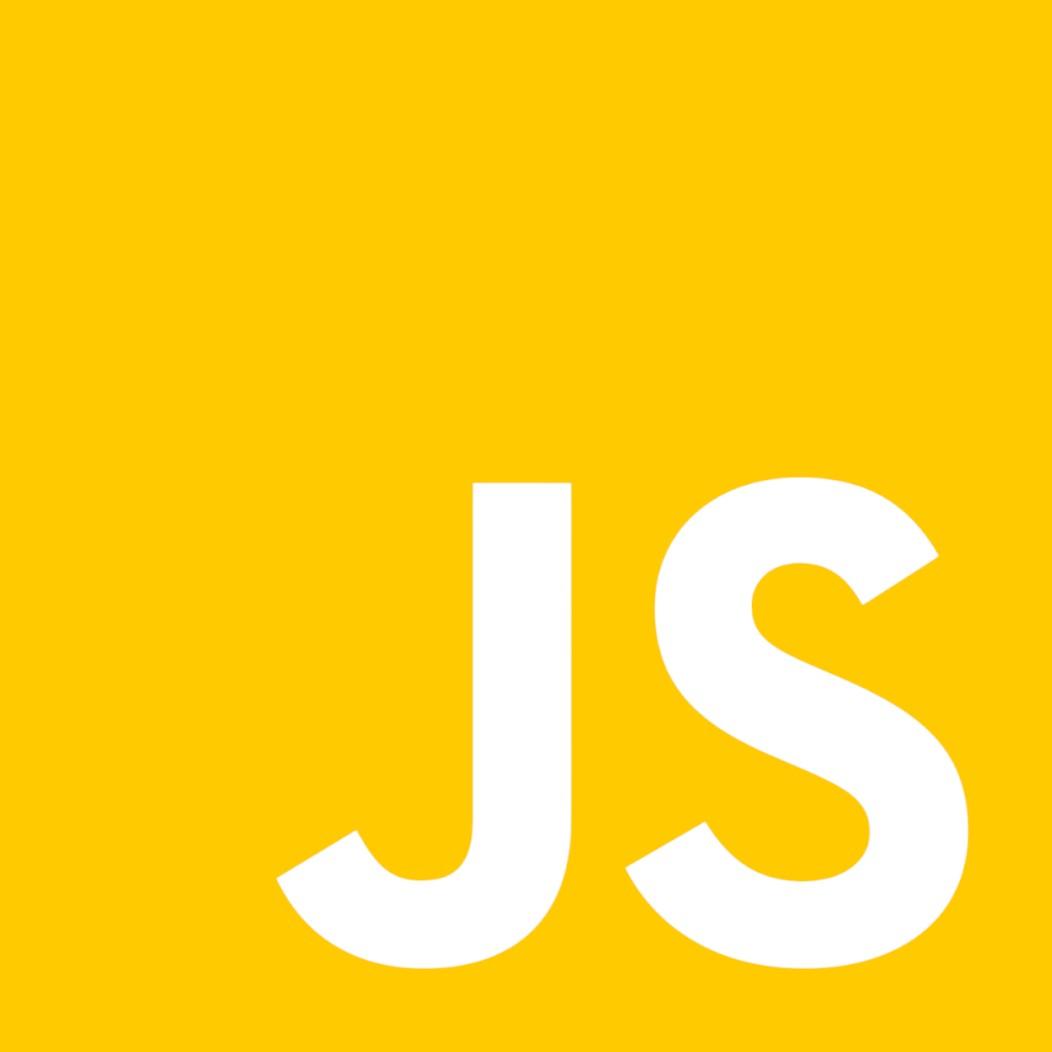
Dado un nodo del DOM, podemos ir a sus inmediatos vecinos utilizando las propiedades de navegación.

Hay dos conjuntos principales de ellas:

* **Para todos los nodos:** *parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling.*
* **Para los nodos elementos:** *parentElement, children, firstElementChild, lastElementChild, previousElementSibling, nextElementSibling.*

Algunos tipos de elementos del DOM, por ejemplo las tablas, proveen propiedades adicionales y colecciones para acceder a su contenido.

Resumen

****

**APARTADO EXTRA: RESUMEN**

# **1 ENTORNO DEL NAVEGADOR Y ESPECIFICACIONES**

**La especificación del DOM**

Describe la estructura del documento, las manipulaciones y los eventos.

Para más información: <https://dom.spec.whatwg.org>

**La especificación del CSSOM**

Describe las hojas de estilo y las reglas de estilo, las manipulaciones con ellas y su vínculo a los documentos.

Para más información: <https://www.w3.org/TR/cssom-1/>

**La especificación del HTML**

Describe el lenguaje HTML (por ejemplo, etiquetas), y también el BOM (modelo de objeto del navegador) que describe varias funciones del navegador como *setTimeout*, *alert* o *location*. Esta toma la especificación DOM y la extiende con muchas propiedades y métodos adicionales.

Para más información: <https://html.spec.whatwg.org>

# **2 ÁRBOL DEL MODELO DE OBJETOS DEL DOCUMENTO (DOM)**

Un documento HTML/XML está representado dentro del navegador como un árbol de nodos (DOM).

* Las etiquetas se convierten en nodos de elemento y forman la estructura.
* Los textos se convierten en nodos de texto.
* Todos los elementos de HTML tienen su lugar en el DOM, incluso los comentarios.

Podemos utilizar las herramientas para desarrolladores para inspeccionar el DOM y modificarlo manualmente.

Para más información: <https://developers.google.com/web/tools/chrome-devtools>

Los nodos del DOM tienen propiedades y métodos que nos permiten desplazarnos entre ellos, modificarlos, movernos por la página, y más.

# **3 RECORRIENDO EL DOM**

Un documento HTML/XML está representado dentro del navegador como un árbol de nodos (DOM).

* Las etiquetas se convierten en nodos de elemento y forman la estructura.
* Los textos se convierten en nodos de texto.