

El navegador: más allá de la visualización de páginas web

"Los datos son el petróleo del siglo XXI. El despliegue de sensores y el incremento de la capacidad del procesamiento, son claves en la transformación de muchos sectores y en la creación de un mundo más medible y programable."

César Alierta – Empresario y abogado



Photo by [Agefis](#) on [Unsplash](#)

Como sabes, Javascript nació como un lenguaje ligado a la web, y una de sus principales funciones consiste en aportarle interacción y dinamismo a sitios web. Hoy en día se ha convertido en uno de los lenguajes de programación más populares, gracias a su versatilidad y múltiples usos. A Javascript lo puedes encontrar en la programación de Front-end, como hemos estado trabajando hasta el momento, pero también lo puedes utilizar para el desarrollo de aplicaciones en el Back-end con [Node JS](#). ¡Incluso puedes encontrar Javascript en la [robótica](#) o en el desarrollo de [aplicaciones para subir a los stores](#) como Google Play, Apple Store o Windows Store!

Encuentra en [este video](#) los múltiples usos de Javascript.



Por el momento nos enfocaremos en el uso de Javascript para el Front-End. El navegador es una herramienta muy útil tanto **si eres usuario/a como developer**. Lo utilizamos para las actividades diarias como mantenernos informados, estudiar, entretenernos, comunicarnos, y ¡hasta para pagar los servicios! Como developers además nos brinda una serie de recursos que resultan fundamentales para desempeñar nuestro trabajo. En esta bitácora encontrarás cuáles son y cómo aplicarlos.

Qué sucede cuando se unen HTML y Javascript

Hasta ahora, sabes que Javascript es el lenguaje de programación orientado a objetos, y conociste a las funciones en la bitácora anterior, una de sus características más importantes. Si retomamos lo que aprendiste de HTML y CSS verás cómo vincular estas dos tecnologías con Javascript.

- **Los documentos HTML permiten incluir fragmentos de código Javascript.** Dentro de un documento HTML puede haber ninguno, uno o varios scripts de JavaScript. Sin embargo, si quisieras incluir programas más largos, escribirlos directamente dentro del HTML sería engorroso. Es por eso que existe el [atributo src](#), mediante el cual puedes indicar un archivo externo para utilizar.

Recuerda que el código HTML se encarga de mostrar en la pantalla el contenido. Vamos a profundizar un poco más lo que significa esa 'pantalla' para nosotros/as.

- **Cuando abrimos una página web en el navegador, este recupera el texto HTML de la página y lo analiza.** Genera un modelo que es lo que vemos en la pantalla de la computadora: esto es lo que llamamos [DOM](#), es una abreviatura de [Document Object Model](#).

DOM es la estructura de objetos que genera el navegador cuando se carga un documento.

Según [Uniwebciudad](#): “La creación del Document Object Model o DOM es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas”.



Sobre el DOM

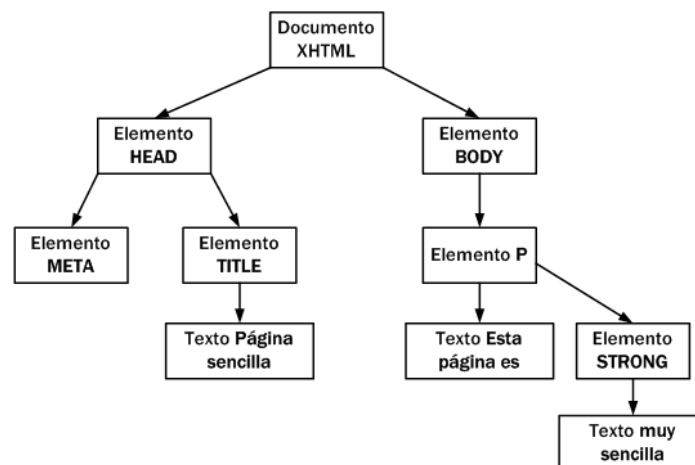
El DOM tiene un potencial enorme, ya que te permitirá a través del navegador **¡modificar la página web en tiempo real!** Podrás intervenir sobre el contenido, el tamaño, el color, las posiciones, e incluso podrás hacer aparecer y desaparecer elementos. Gracias al DOM, verás el resultado en la misma pantalla (claro que si el sitio no es tuyo o no tienes acceso al código fuente, las ediciones que hagan solo se reflejarán en tu pantalla pero a nivel local).

¿Estás emocionado/a por empezar? ¡Todavía queda información relevante que debes tener en cuenta!

DOM transforma todos los documentos HTML en un conjunto de nodos, que están interconectados y que representan los contenidos de las páginas web de una manera sencilla para poder ser utilizarlos. Un **nodo** es cada uno de los elementos de una lista enlazada. A continuación te mostramos cómo se representa.

Imagina la estructura del DOM como la de un árbol con un tronco, ramas, otras ramas que salen de ellas y hojas al final de cada una. La unión de todos los nodos es conocida como **árbol de nodos**.

Gráficamente se vería así:

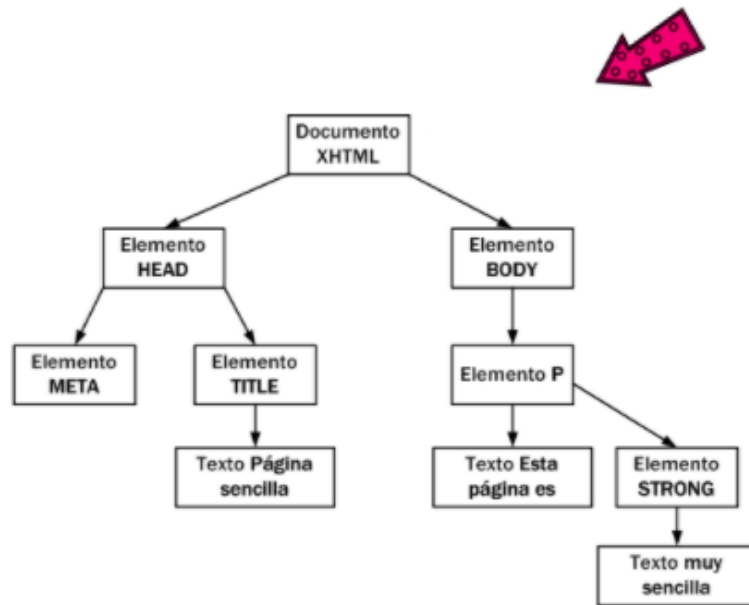


Observa en el gráfico que estos nodos tienen un **parentesco** entre sí: los elementos que contienen a otros son los nodos **padres**, mientras que los elementos contenidos son **hijos**.

Mediante el DOM, este modelo que representa a nuestro documento HTML, podemos “navegar” por la estructura, acceder a sus partes e interactuar

con ellas. Todo esto, sin necesariamente saber el identificador del elemento que estamos buscando.

Existen distintas maneras de hacerlo. Por ejemplo, la palabra reservada `document` te permite acceder al nodo principal



`Document` nos proporciona una serie de funciones para buscar nodos, podremos acceder a un o varios nodos de la siguiente manera:

- **Por elemento.** La función `getElementsByName` nos devuelve un array con todos los nodos encontrados que sean igual al tag que envió como parámetro.

```
var by_tag = document.getElementsByTagName("div");
```

- **Por ID.** La función `getElementById` nos permite buscar un nodo a través del atributo id y retorna directamente el nodo encontrado.

```
var by_id = document.getElementById('header');
```

- **Por Clase.** La función `getElementsByClassName` te devolverá un array con todos los nodos encontrados que sean iguales al parámetro que envíes.

```
var classes = document.getElementsByClassName("bg_red");
```

- **Por Query.** La función `querySelector` a través de su parámetro te permite buscar nodos con la misma sintaxis que utilizas en CSS y te devolverá directamente el nodo. Atención, si tu selector encuentra más de un nodo solamente te devolverá **el primer nodo** encontrado.

```
var buscar = document.querySelector(".bg_red");
```

¡Encuentra [aquí](#) los 4 ejemplos en funcionamiento!

Una vez que accedes al nodo deseado, utiliza la variable para poner en funcionamiento algunas propiedades y métodos disponibles. Utilicemos por ejemplo una variable llamada `element` para ver las propiedades, podría ser el resultado de alguno de los selectores que recientemente detallamos. Puedes encontrar un listado muy útil aquí:

Listado: Propiedades

Propiedades

- `element.attributes` contiene un array de atributos.
- `element.className` devuelve o establece el valor del atributo class
- `element.classList` devuelve un array atributos de clase del elemento y permite trabajar con él.
- `element.firstChild` devuelve el siguiente nodo.
- `element.id` devuelve o establece el valor del atributo id
- `element.innerHTML` devuelve o establece la sintaxis HTML
- `element.lastElementChild` devuelve el último nodo
- `element.name` obtiene o establece el atributo del nombre. Sólo aplica a los siguientes elementos:

`<a>`,



```

<applet>
<button>
<form>
<frame>
<iframe>
<img>
<input>
<map>
<meta>
<object>
<param>
<select>
<textarea>.

```

- `element.outerHTML` **obtiene o establece el fragmento HTML completo, incluyendo el propio tag.**
- `element.tagName` **devuelve el nombre del elemento.**

Es importante que tengas en cuenta que debes estar posicionado/a sobre el nodo para poder acceder a sus propiedades.

Por ejemplo, si utilizas una función que devuelve más de un nodo, te devolverá un array. Si en ese momento intentas acceder a una propiedad te arrojará un error. Deberás acceder al ítem en tu array para tener disponibles las propiedades.

En el primer ejemplo del codepen buscamos todos los `divs` del documento y nos devuelve un array con 5 `divs`.

```

var by_tag = document.getElementsByTagName("div");
console.log(by_tag.innerHTML);
//producirá undefined. by_tag es un array y no posee
la propiedad innerHTML
console.log(by_tag[2].innerHTML);
//producirá un 3 (recuerda que la primera posición del
array es 0)

```



¡A modificar el DOM!

Como sabes, uno de los aspectos más poderosos e interesantes a la hora de agregar Javascript a tus sitios es que eso te permite hacerlos interactivos. Así que, ¡buenas noticias! ¡Llegamos al momento más esperado!

Al trabajar con el DOM, no solo podrás obtener información de él, sino además modificarlo dinámicamente. Igual que para acceder a los elementos, existen diversas maneras de modificar el DOM.

La manera más sencilla es a través de las **propiedades: puedes acceder a cualquiera de ellas y cambiar su valor**. Por ejemplo, si modificas la propiedad `innerHTML` no solo estarás cambiando la propiedad, sino también lo que el usuario ve por pantalla:

```
//Ejemplo con este código
//<div id="msj">Hola a todos</div>

//El usuario en este punto visualiza "Hola a todos"
msj = document.getElementById("msj")
msj.innerHTML = "Estoy cambiando";
//El usuario visualiza "Estoy cambiando"
```

También puedes modificar nodos existentes, crear nuevos o incluso crear nuevos nodos dentro de otros nodos:

- **Crear nuevos nodos:**

`document.createElement(tagName)` : recibe como parámetro el tipo de tag que deseamos crear y devuelve un objeto con el nodo creado .

```
nuevo_nodo = document.createElement("span")
```

- **Eliminar nodo:**

`element.remove()` : para eliminar el nodo actual del árbol.

```
var parrafo = document.getElementById("p");
parrafo.remove();
```



- **Reemplazar nodos:**

`element.replaceWith(nuevoNodo)`: para reemplazar el nodo seleccionado por el nodo enviado por parámetro.

```
nuevo_nodo = document.createElement("span")
var parrafo = document.getElementById("p");
parrafo.replaceWith(nuevo_nodo);
//reemplaza el <p> por un <span>
```

- **Insertar nodos:**

`element.appendChild`: para agregar un nodo al final de la lista de nodos hijos.

```
//<div id="p">hola</div>
nuevo_nodo = document.createElement("p")
var parrafo = document.getElementById("p");
parrafo.appendChild(nuevo_nodo);
//Resultado: <div id="p">hola<p></p></div>
```

`element.insertBefore`: si necesitas incluir un nodo justo antes del nodo indicado.

```
//<div id="p">hola</div>
nuevo_nodo = document.createElement("p")
var parrafo = document.getElementById("p");
parrafo.insertBefore(nuevo_nodo);
//Resultado: <p><div id="p">hola</div></p>
```

Por supuesto que existen más métodos. Al igual que las propiedades puedes encontrarlos en la [web](#) de Mozilla.

Los datos se guardan en el navegador

Este fue el punto de partida de la bitácora, ¿cierto?: **el navegador está presente en nuestras vidas de muchas maneras.** Tal como señala la web de [significados](#), su función es:



“posibilitar al usuario la visualización de páginas web y todos sus componentes: documentos, texto, imágenes, videos, audios, hipervínculos, etc. A través del navegador, el usuario puede realizar múltiples actividades: enviar y recibir correos electrónicos, acceder a páginas web y redes sociales, seleccionar y guardar sus páginas favoritas, imprimir documentos, mantener registros de su actividad, almacenar información en la nube, instalar aplicaciones, etc.”

Muchas de esas situaciones tienen lugar y se repiten cotidianamente. Sería engorroso, por ejemplo, tener que ingresar nuestro usuario y contraseña cada vez que entramos a revisar los mails o redes sociales. Pero, ¿qué sucede por detrás para que sea posible que los datos permanezcan guardados?

JavaScript posee diferentes funciones respecto a la información:

- **Guardar información de forma permanente.** Una funcionalidad en conjunto con los navegadores es [LocalStorage](#) para poder almacenar información aun después de cerrar el navegador. No se elimina hasta que el/la usuario/a o la aplicación lo determine.

Para guardar información utilizarás `setItem` donde definirás un nombre y un valor.

Mediante `getItem` más el nombre que has definido, obtendrás dicha información.

`removeItem` más el nombre te permitirá eliminar la información de forma permanente.

```
localStorage.setItem("nombre", "Jimena");
var traer_datos = localStorage.getItem("nombre");
console.log(traer_datos); //imprime Jimena
//En este punto puedo reiniciar el navegador que
siempre va a traer el valor que haya almacenado
localStorage.removeItem("nombre"); //eliminó
definitivamente "Jimena"
```

- **Guardar información momentáneamente.** Existe también una funcionalidad llamada [sessionStorage](#), que funciona de un modo muy similar a LocalStorage. La sintaxis es idéntica de hecho, la diferencia es que la información de SessionStorage se elimina en cuanto el/la usuario/a abandona la página, ya sea dirigiéndose a otro sitio o cerrando el navegador. Mientras tanto, la información guardada por LocalStorage va a



mantenerse indefinidamente. Tiene los mismos métodos que localStorage, solo que la duración de lo que guardemos en este storage será equivalente a la duración de la sesión.

```
sessionStorage.setItem("nombre", "Jimena");  
var traer_datos = sessionStorage.getItem("nombre");  
console.log(traer_datos); //imprime Jimena  
//Si reinicio el navegador en este punto pierdo el  
valor "Jimena"  
sessionStorage.removeItem("nombre");           //eliminó  
definitivamente "Jimena" de la sesión
```



Siempre hay que tener cuidado con la información que guardas, ¡nunca sabes quién puede estar teniendo acceso a ella! Por eso, una buena práctica es no guardar datos sensibles como contraseñas importantes, datos de tarjetas de crédito, etc.

Resumiendo

Está claro que el navegador web es muy importante tanto como usuarios/as como developers y, en especial, para el manejo de JavaScript. Aprendiste cómo se relaciona con los otros lenguajes, cómo se utiliza para modificar las páginas web y manipular los datos que se guardan. ¿Te animas a hacer los primeros cambios en tu web favorita (¡o en aquella que detestes!) y probarte a tí mismo que puedes generar cambios interesantes?


¡Prepárate para el próximo encuentro!

Profundiza

-  [Understanding the DOM](#)
-  [LocalStorage](#) y [SessionStorage](#)

Herramientas




 La consola de los navegadores y el DOM son mejores amigos. Ten presente la consola de desarrollo todo el tiempo, ya que te será de mucha utilidad ver el comportamiento de los elementos a medida que los vas modificando.


Comunidad

[Meetup](#) es una aplicación que reúne miles de eventos de tecnología al año ¡y puedes encontrar muchos gratuitos para participar!

Allí encontrarás que muchos expertos/as del mundo forman grupos que organizan sus propios encuentros y publican información dentro su comunidad.

 Encuentra grupos de Javascript para compartir en el próximo encuentro y si te animas, ¡súmate a ellos! Siempre están publicando información de mucho valor.

Challenge

 Toma cualquier desarrollo que hayas hecho hasta el momento y modifica algún elemento a través del DOM.

En el próximo encuentro nos contarás qué modificación realizaste.

 ¡No olvides traer maquettato el wireframe que te mostramos en clase!

Potencia tu Talento - LinkedIn

Tu perfil de LinkedIn

Tu perfil de LinkedIn es la principal herramienta para darte a conocer en el ecosistema profesional, tener un perfil completo puede hacer que los *recruiters* se interesen en tu perfil, el algoritmo te posicione mejor, y consigas las mejores oportunidades de empleo.

Puede que no estés buscando activamente un cambio de trabajo, pero tener tu información actualizada hará que nuevas oportunidades puedan surgir aunque no las busques. Debajo tienes un link a un artículo que recorre paso a paso las secciones del perfil para que tengas todos los tips para completarlo.

Además, LinkedIn es un gran espacio para dar a conocer tu portfolio y compartir contenido con tus colegas.

Valida tu conocimiento



¿Sabías que en LinkedIn puedes realizar tests rápidos de 15 minutos para validar tu conocimiento sobre un tema? Puedes validar aptitudes como Git, Adobe Illustrator, Javascript o Python. Encuentra esta opción en la sección "Aptitudes y validaciones".

 [Tu perfil de LinkedIn paso a paso](#)

 [LinkedIn: mucho más que una red para buscar trabajo](#)

 [La foto perfecta para tu CV](#)

