

## Contenido

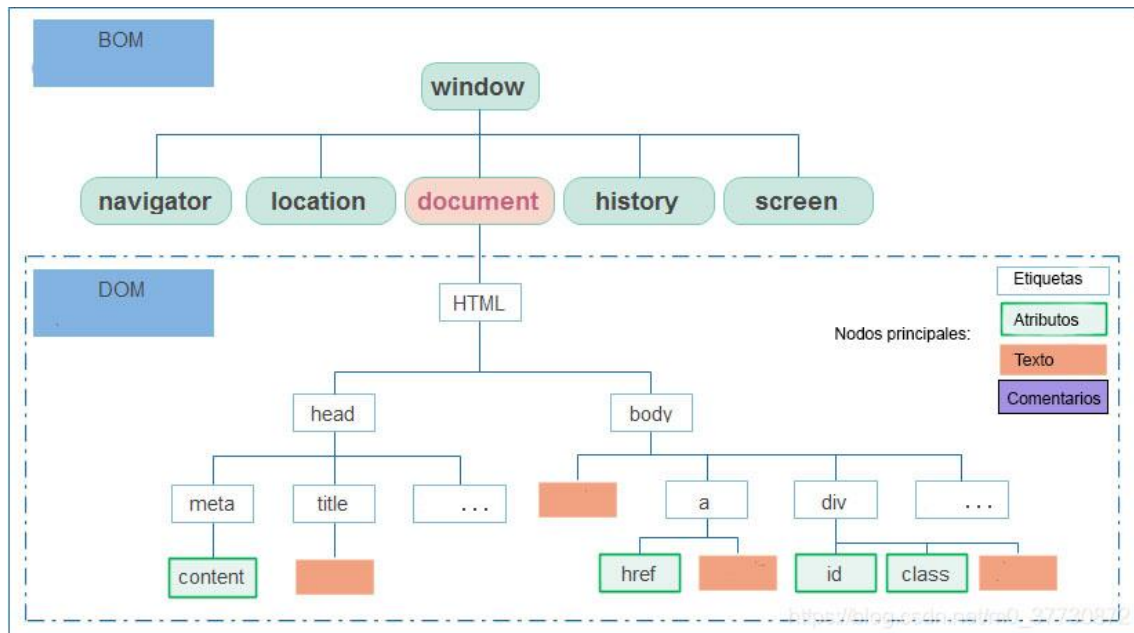
Contenido .....	1
1 EL OBJETO WINDOW .....	2
1.1 BOM, MODELO DE OBJETOS DEL NAVEGADOR .....	2
1.2 EL OBJETO NAVIGATOR .....	2
1.3 OBJETO SCREEN.....	3
1.4 OBJETO LOCATION .....	3
1.5 OBJETO HISTORY .....	4
1.6 OTRAS PROPIEDADES Y MÉTODOS DE WINDOW.....	4
2 ¿QUÉ ES EL DOM? .....	6
2.1 EL MODELO DE OBJETOS DEL DOCUMENTO.....	6
2.2 EL OBJETO DOCUMENT .....	7
2.3 TIPOS DE NODOS (NODE) .....	7
3 API NATIVA DE JAVASCRIPT.....	8
4 LIBRERÍAS DE TERCEROS.....	9

Cuando comenzamos en el mundo del **desarrollo web**, normalmente comenzamos por aprender a escribir etiquetado o **marcado HTML** y además, añadir **estilos CSS** para darle color, forma y algo de interacción. Sin embargo, a medida que avanzamos, nos damos cuenta que en cierta forma podemos estar bastante limitados.

Si únicamente utilizamos HTML/CSS, sólo podremos crear **páginas «estáticas»** (*sin demasiada personalización por parte del usuario*), pero si añadimos Javascript, podremos crear **páginas «dinámicas»**. Cuando hablamos de páginas dinámicas, nos referimos a que podemos dotar de la potencia y flexibilidad que nos da un lenguaje de programación para crear documentos y páginas mucho más ricas, que brinden una experiencia más completa y con el que se puedan automatizar un gran abanico de tareas y acciones.

## 1 EL OBJETO WINDOW

### 1.1 BOM, MODELO DE OBJETOS DEL NAVEGADOR



El objeto Windows es la propia ventana y es el elemento fundamental del manejo de las aplicaciones web desde JavaScript. Es también la raíz de toda la organización de objetos a los que JavaScript puede acceder para manipular todos los aspectos de una aplicación web. Por ejemplo, en el caso de **alert** (método que muestra un cuadro de diálogo) la sintaxis completa es **window.alert()** aunque casi todos los desarrolladores utilizemos **alert()** a secas.

Una característica fundamental de JavaScript, de hoy en día, es que se puede ejecutar en todo tipo de entornos, no solo en los navegadores. Por ejemplo, Node.js no dispone del objeto Window porque es un entorno basado en la consola del navegador. Cuando usamos este objeto es señal de que estamos desarrollando una aplicación del lado del cliente.

El **BOM** es el modelo de objetos del navegador y aglutina toda la estructura organizativa de los objetos del navegador. A partir de ahora **el navegador no es una simple pantalla, es un conjunto de objetos que podemos consultar y manipular**. En él tenemos objetos como el navegador, la barra de búsqueda, la consola, el documento, etc.

En este apartado, vamos a conocer alguno de los objetos importantes del navegador y veremos una gran lista de métodos y propiedades de todos ellos.

### 1.2 EL OBJETO NAVIGATOR

Representa al navegador del usuario, posee numerosas propiedades de lectura que nos permiten obtener información interesante, quizá la más importante es **userAgent** que permite obtener la cadena de información del navegador, usuario, motor....

Otras propiedades y métodos son:

```
console.log(navigator.userAgent);
```

PROPIEDAD O METODO	USO
<b>clipboard</b>	Devuelve un objeto para gestionar desde el código el portapapeles
<b>cookieEnabled</b>	Valor booleano que indica si las cookies están activadas en el navegador.
<b>geolocation</b>	Obtiene un objeto que permite usar la API de geolocalización para acceder a la posición GPS del usuario.
<b>javaEnabled()</b>	Devuelve un valor booleano que indica si el plugin de Java está activado en el navegador
<b>Language</b>	Devuelve un string con el código de lenguaje del navegador del usuario. Por ejemplo "es-ES"
<b>mimeType</b>	Obtiene un array con los tipos MIME aceptados por el navegador
<b>online</b>	Valor booleano que indica si el navegador está trabajando on line
<b>plugins</b>	Devuelve un array con información sobre los plugins instalados en el navegador.
<b>serviceWorker</b>	Obtiene un objeto de tipo ServiceWorkerContainer capaz de trabajar con objetos de tipo ServiceWorker
<b>storage</b>	Devuelve un objeto de tipo StorageManager capaz de manejar la API de almacenamiento de datos persistentes

### 1.3 OBJETO SCREEN

Esta propiedad nos permite acceder a un objeto que representa la pantalla en la que está navegando el usuario. Propiedades interesantes de este objeto son:

PROPIEDAD O METODO	USO
<b>availTop</b>	Primera coordenada superior de la pantalla que se puede utilizar en nuestras aplicaciones.
<b>availLeft</b>	Primera coordenada izquierda de la pantalla que se puede utilizar en nuestras aplicaciones.
<b>availHeight</b>	Número que indica el tamaño en píxeles de la altura de pantalla del usuario
<b>availWidth</b>	Número que indica el tamaño en píxeles de la anchura de pantalla del usuario
<b>height</b>	Altura completa de la pantalla en píxeles
<b>width</b>	Anchura completa de la pantalla en píxeles
<b>ColorDepth</b>	Devuelve el número <b>n</b> de colores de la pantalla. Se calculan elevando dos al número devuelto: 2 <sup>n</sup>
<b>orientation</b>	Orientación de la pantalla

### 1.4 OBJETO LOCATION

Objeto que representa la URL de la aplicación. Está propiedad también es accesible mediante **document.location**, que es una referencia al mismo objeto.

Podría devolver algo como: <https://www.castillopuche.edu/>

Más interesante es el hecho de que podemos ir a otra página simplemente cambiando el valor de la propiedad:

```
location.href=https://github.com;
```

Este objeto tiene otra serie de propiedades (que son modificables) y que nos permiten obtener cada apartado URL de la página:

PROPIEDAD O METODO	USO
<b>protocol</b>	Obtiene el nombre del protocolo que se utiliza para acceder a la aplicación. Ejemplo: <b>https</b> :
<b>host</b>	Toma de la URL lo que se corresponde con la dirección del host. Incluye el puerto si no es un puerto estándar.  host  Ejemplo:www.castillopuche.edu:3241
<b>hostname</b>	Idéntica al anterior, pero sin incluir el puerto.
<b>pathname</b>	Obtiene la ruta de directorios a partir del host en la URL. Por ejemplo:/manuales/html/introduccion-html.html
<b>search</b>	Obtiene la cadena de búsqueda de la URL. Por ejemplo: ?lang=es&registrado=si
<b>hash</b>	Obtiene el marcador, si es que lo lleva, de la URL. Por ejemplo en la dirección http://www.castillopuche.edu/web#lmsgi Obtendría <b>#lmsgi</b>
<b>username</b>	Obtiene el nombre de usuario que haya en la URL, si es que lo hay.
<b>password</b>	Si es el caso, devuelve la contraseña que haya en la URL.
<b>origin</b>	Es la única propiedad no modificable, es de solo lectura. Permite obtener de la URL la dirección canónica. La cual está formada solo por el protocolo, el puerto y el nombre de host.
<b>reload</b>	carga la página. El parámetro opcional servidor es un booleano que, si vale true, obliga a recargar la página desde el servidor, si vale false se permite cargar la página usando la caché del navegador.

## 1.5 OBJETO HISTORY

Se trata del objeto que representa el historial de páginas visitadas por el usuario. El método más interesante es **go**, el cual recibe un número que nos permite navegar por el historial de páginas: Así: **go(-1)** iría a la página anterior en el historial, **go(1)** iría a la página siguiente, **go(0)** es otra forma de recargar la página actual.

La otra propiedad interesante de este objeto es **length**, que devuelve el tamaño actual del historial. Ejemplo:

```
history.go(-(history.length-1)); //volvemos a la primera página del historial
```

## 1.6 OTRAS PROPIEDADES Y MÉTODOS DE WINDOW

El objeto **window** tiene otras propiedades y métodos que pueden ayudarnos a realizar acciones desde JavaScript. Ya conocemos las propiedades **history**, **screen** y **navigator** cuyas posibilidades hemos comentado. Hay otros objetos interesantes.

También conocemos métodos de window, concretamente **alert**, **prompt** y **confirm** que se encargan de los mensajes al usuario.

A continuación, se detallan algunas de esas propiedades o métodos.

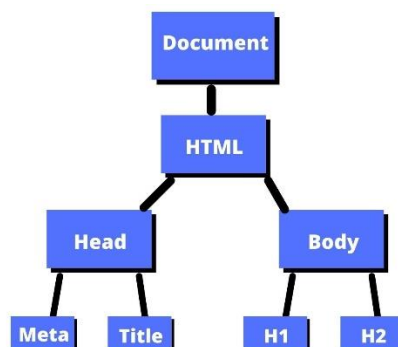
PROPIEDAD O METODO	USO
<b>innerWidth</b>	Anchura interior de la ventana. Es la anchura real disponible en el área de contenido de la ventana teniendo en cuenta su tamaño actual
<b>innerHeight</b>	Altura interior de la ventana. Es la altura real disponible en el área de contenido de la ventana teniendo en cuenta su tamaño actual
<b>outerWidth</b>	Anchura exterior de la ventana
<b>outerHeight</b>	Altura exterior de la ventana.
<b>screenX</b>	Distancia de la ventana al borde izquierdo de la pantalla.
<b>screenLeft</b>	Idéntica a la anterior
<b>screenY</b>	Distancia de la ventana al borde superior de la pantalla.
<b>screenTop</b>	Idéntica a la anterior
<b>scrollX</b>	Indica el desplazamiento horizontal que el usuario ha realizado usando las barras de desplazamiento horizontales o arrastrando un dispositivo táctil en horizontal.
<b>scrollY</b>	Indica el desplazamiento vertical que el usuario ha realizado usando las barras de desplazamiento verticales o arrastrando un dispositivo táctil en vertical.
<b>status</b>	Referencia a la barra de estado de la ventana. Permite ver y modificar su contenido. La barra de estado era una franja horizontal situada en la parte inferior de la ventana en la que se daba información de estado. Hoy en día no se utiliza esta barra, pero la propiedad sigue estando disponible.
<b>console</b>	Consola del navegador. Accede a todos los métodos de la consola, incluido console.log.
<b>event</b>	Objeto del evento actualmente en ejecución. Los eventos protagonizan la siguiente unidad.
<b>fullScreen</b>	Booleano que indica si la aplicación se muestra a pantalla completa
<b>localStorage</b>	Referencia al objeto de tipo <b>Storage</b> que permite almacenamiento de datos en modo local sin tiempo de expiración. Lo veremos más adelante.
<b>getSelection()</b>	Obtiene un objeto del tipo <b>Selection</b> que nos sirve para obtener información del texto seleccionado. Si queremos simplemente obtener el texto seleccionado por consola, sería; <code>console.log(getSelection().toString());</code>
<b>scroll(x,y)</b> o bien <b>scroll(objetoscroll)</b>	<b>scroll(0,100); //le pasamos las coordenadas</b> Se puede pasar en lugar de las coordenadas un objeto <b>ScrollToOptions</b> con las siguientes propiedades: <ul style="list-style-type: none"><li>• Top</li><li>• Left</li><li>• Behavior: auto smooth ( se desliza sin animación o con animación suave). Ejemplo: <code>Scroll({</code></li></ul>

PROPIEDAD O METODO	USO
	top:1000, left:0, behavior: "smooth" });
<b>scrollby(x,y)</b> o bien <b>scrollby(objetoscroll)</b>	Los parámetros funcionan igual que la propiedad anterior, pero ahora el desplazamiento es relativo a la posición actual. Ejemplo: scrollBy({ top:innerHeight, left:0, behavior: "smooth" }); Este código hace que la página se desplace suavemente una pantalla completa hacia abajo.
<b>scrollto(x,y)</b> o bien <b>scrollto(objetoscroll)</b>	Mismos parámetros pero los valores son coordenadas, no pixeles (normalmente coinciden)
<b>stop()</b>	Detiene la carga de la página
<b>Find(texto)</b>	Busca el texto en la página actual, devuelve booleano y la mayoría de navegadores hacen scroll hasta posicionarse sobre el y lo resaltan.
<b>getCompujetStyle(elem ento)</b>	<b>Permite obtener la información de las propiedades CSS en uso por parte de un elemento.</b>
<b>Open(URL,nombreVent ana,ajustes)</b>	Abre una nueva ventana con el contenido y ajustes indicados

## 2 ¿QUÉ ES EL DOM?

### 2.1 EL MODELO DE OBJETOS DEL DOCUMENTO

Las siglas **DOM** significan **Document Object Model**, o lo que es lo mismo, la estructura del documento HTML. Una página HTML está formada por múltiples etiquetas HTML, anidadas una dentro de otra, formando un árbol de etiquetas relacionadas entre sí, que se denomina **árbol DOM** (o simplemente *DOM*).



En Javascript, cuando nos referimos al **DOM** nos referimos a esta estructura, que podemos modificar de forma dinámica, añadiendo nuevas etiquetas, modificando o

eliminando otras, cambiando sus atributos HTML, añadiendo clases, cambiando el contenido de texto, etc...

Al estar "amparado" por un **lenguaje de programación**, todas estas tareas se pueden automatizar, incluso indicando que se realicen cuando el usuario haga acciones determinadas, como por ejemplo: pulsar un botón, mover el ratón, hacer click en una parte del documento, escribir un texto, etc...

## 2.2 EL OBJETO DOCUMENT

En Javascript, la forma de acceder al DOM es a través de un objeto llamado precisamente **document**, que representa el árbol DOM de la página o pestaña del navegador donde nos encontramos. Este objeto es en realidad una propiedad del objeto **window**. Podemos indicar simplemente la palabra **document** aunque su nombre completo es window. Document. Los diferentes métodos y propiedades del objeto document nos van a permitir acceder a los elementos del documento para su manipulación.

En su interior pueden existir varios tipos de elementos, pero principalmente serán **element** o **node** :

## 2.3 TIPOS DE NODOS (NODE)

Más adelante veremos muchos métodos para acceder a los elementos de un documento, tomemos como ejemplo el método **getElementById** del objeto **document**.

```
let seccion=document.getelementbyid("#s1");
```

Este código recupera un nodo con el id especificado. Todos los nodos poseen una propiedad llamada NodeType. Si la mostramos en este caso:

```
console.log(seccion.nodeType);
```

Se muestra el valor 1. Significa que el node es un elemento. Si ejecutamos este otro código:

```
console.log(document.nodeType);
```

Escribirá el valor 9.

La lista de posibles valores **nodeType** es la siguiente:

VALOR	TIPO DE NODO
1	Elemento
2	Atributo
3	Texto
4	Apartado CDATA
5	Referencia a entidad
6	Entidad
7	Instrucción de procesado
8	Comentario
9	Documento completo
10	Nodo de tipo de documento
11	Nodo de fragmento de código
12	Nodo de anotación

Algunas entradas están muy relacionadas con XNML por lo que no nos serán útiles para manipular documentos HTML, normalmente nos bastará con las tres primeras






Además, Todos los **elementos HTML**, tendrán un tipo de dato específico. Algunos ejemplos:

Tipo de dato	Tipo específico	Etiqueta	Descripción
ELEMENT HTMLElement	HTMLDivElement	<div>	Capa divisoria invisible (en bloque).
ELEMENT HTMLElement	HTMLSpanElement	<span>	Capa divisoria invisible (en línea).
ELEMENT HTMLElement	HTMLImageElement	<img>	Imagen.
ELEMENT HTMLElement	HTMLAudioElement	<audio>	Contenedor de audio.

Obviamente, existen muchos tipos de datos específicos, uno por cada etiqueta HTML.

### 3 API NATIVA DE JAVASCRIPT

En los siguientes capítulos veremos que **JavaScript** nos proporciona un conjunto de herramientas para trabajar de forma nativa con el DOM de la página, entre las que se encuentran:

Capítulo del DOM	Descripción
 <b>Buscar etiquetas</b>	Familia de métodos entre los que se encuentran funciones como <code>.getElementById()</code> , <code>.querySelector()</code> o <code>.querySelectorAll()</code> , entre otras.
 <b>Crear etiquetas</b>	Una serie de métodos y consejos para crear elementos en la página y trabajar con ellos de forma dinámica.
 <b>Insertar etiquetas</b>	Las mejores formas de añadir elementos al DOM, ya sea utilizando propiedades como <code>.innerHTML</code> o método como <code>.appendChild()</code> , <code>.insertAdjacentHTML()</code> , entre otros.
 <b>Gestión de CSS</b>	Consejos para la utilización de la API <code>.classList</code> de Javascript que nos permite manipular clases CSS desde JS, de modo que podamos añadir, modificar, eliminar clases de CSS de un elemento de una forma práctica y cómoda.
 <b>Navegar por tags</b>	Utilización de una serie de métodos y propiedades que nos permiten «navegar» a través de la jerarquía del DOM, ciéndonos a la estructura del documento y la posición de los elementos en la misma.



## 4 LIBRERÍAS DE TERCEROS

En muchos casos, el rendimiento no es lo suficientemente importante como para justificar trabajar a bajo nivel, por lo que se prefiere utilizar algunas **librerías de terceros** que nos facilitan el trabajo a costa de reducir mínimamente el rendimiento, pero permitiéndonos programar más rápidamente.

Si es tu caso, puedes utilizar alguna de las siguientes librerías para abstraerte del DOM:

Librería	Descripción	GitHub
<a href="#">RE:DOM</a>	Librería para crear interfaces de usuario, basada en DOM.	<a href="#">@redom/redom</a>
<a href="#">Voyeur.js</a>	Pequeña librería para manipular el DOM	<a href="#">@adriancooney/voyeur.js</a>
<a href="#">Htmljs</a>	Motor de renderización de HTML y data binding (MVVM)	<a href="#">@nhanfu/htmljs</a>
<a href="#">DOMtastic</a>	Librería moderna y modular para DOM/Events	<a href="#">@webpro/DOMtastic</a>
<a href="#">UmbrellaJS</a>	Librería para manipular el DOM y eventos	<a href="#">@franciscop/umbrella</a>
<b>SuperDOM</b>	Manipulando DOM como si estuvieras en 2018	<a href="#">@szaranger/superdom</a>

Muchas veces, también se eligen frameworks de Javascript para trabajar, que en cierta forma también te abstraen de tener que gestionar el DOM a bajo nivel, y lo cambian por realizar otras tareas o estrategias relacionadas con el framework escogido.