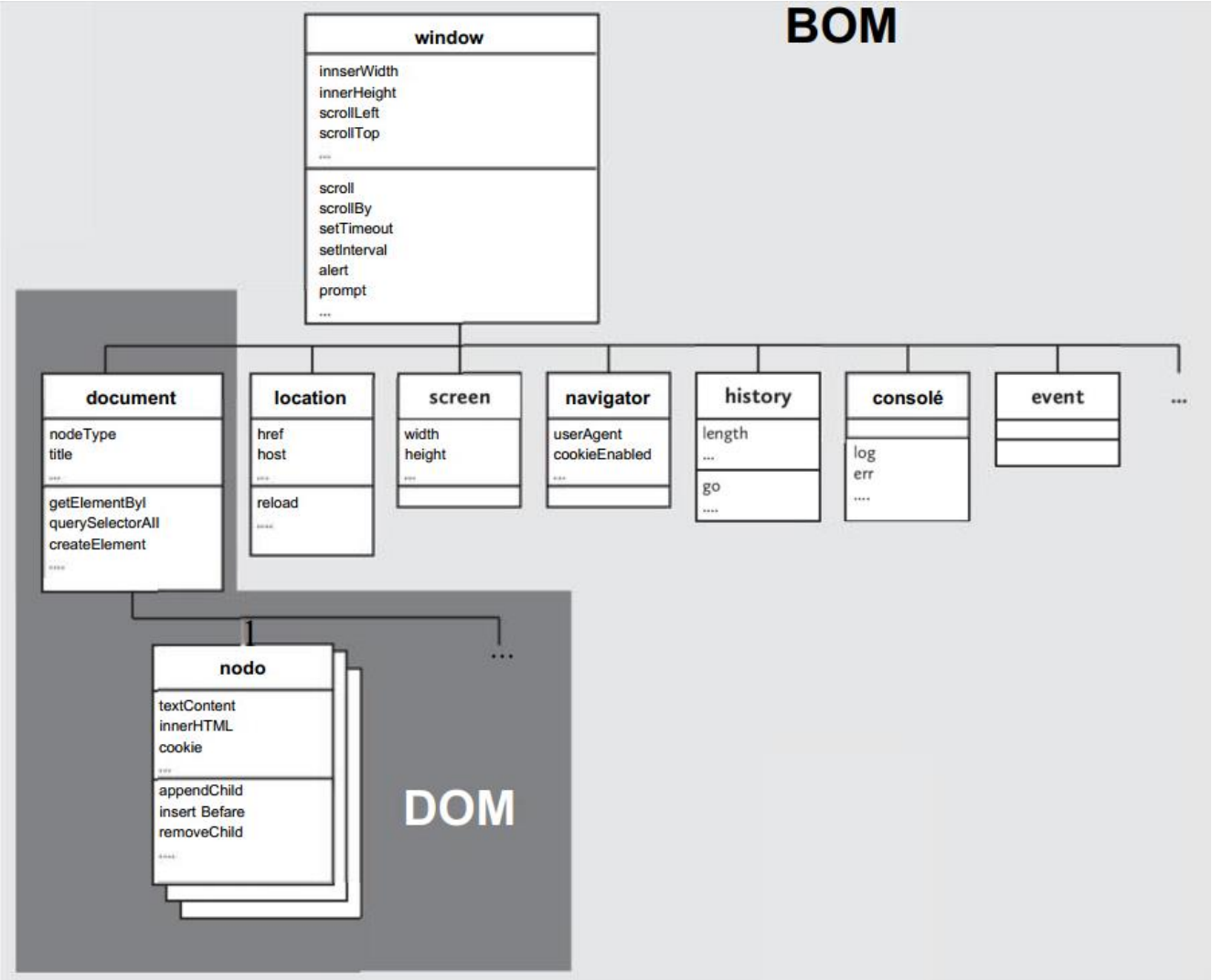
CONTENIDOS

5.1 EL OBJETO WINDOW

5.1.1 BOM, MODELO DE OBJETOS DEL NAVEGADOR

BOM abreviatura de *Browser Objects Model* (***Modelo de Objetos del Navegador***) y que aglutina toda la estructura organizativa de los objetos del navegador. A partir de ahora, el navegador no es una simple pantalla, sino 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.

A continuación se muestra la estructura del modelo de objetos del navegador (BOM), resaltando el modelo de objetos del documento:



Aunque el BOM no forma parte del estándar oficial del lenguaje JavaScript, lo cierto es que los distintos navegadores han igualado la forma de trabajar con estos objetos. Por lo tanto, podemos hablar de un uso estándar de los objetos del navegador.

El objeto window 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.
Hemos trabajado ya con varios métodos del objeto window (**alert, prompt, confirm**..) que resultan tan importantes que incluso se puede obviar su nombre al usar sus métodos y propiedades. Se suelen invocar con el nombre del método sin anteponer el objeto al que pertenecen en lugar que con el nombre completo ***window.alert***, ***window.prompt***..

Nota: Definimos como objeto, una entidad con una serie de propiedades que definen su estado, y unos métodos (funciones), que actúan sobre esas propiedades.

La forma de acceder a una propiedad de un objeto es la siguiente:
nombreobjeto.propiedad
La forma de acceder a un método de un objeto es la siguiente:
nombreobjeto.metodo([parámetros opcionales])

En la jerarquía de objetos, tenemos en la parte superior el objeto window. Este objeto está situado justamente ahí, porque es el contenedor principal de todo el contenido que se visualiza en el navegador. Tan pronto como se abre una ventana (window) en el navegador, incluso aunque no se cargue ningún documento en ella, este objeto window ya estará definido en memoria.

Además de la sección de contenido del objeto window, que es justamente dónde se cargarán los documentos, el campo de influencia de este objeto, abarca también las dimensiones de la ventana, así como todo lo que rodea al área de contenido: las barras de desplazamiento, barra de herramientas, barra de estado, etc.

Atención: en los navegadores más modernos, los usuarios tienen la posibilidad de abrir las páginas tanto en nuevas pestañas dentro de un navegador, como en nuevas ventanas de navegador. Para JavaScript tanto las ventanas de navegador, como las pestañas, son ambos objetos window.

Acceso a propiedades y métodos: Para acceder a las propiedades y métodos del objeto window, lo podremos hacer de diferentes formas, dependiendo más de nuestro estilo, que de requerimientos sintácticos. Así, la forma más lógica y común de realizar esa referencia, incluiría el objeto window tal y como se muestra en este ejemplo:
window.nombrePropiedad
window.nombreMétodo([parámetros])

Los parámetros van entre corchetes, indicando que son opcionales y que dependerán del método al que estemos llamando.

Un objeto window también se podrá referenciar mediante la palabra self, cuando estamos haciendo la referencia desde el propio documento contenido en esa ventana:
self.nombrePropiedad
self.nombreMétodo([parámetros])

Debido a que el objeto window siempre estará presente cuando ejecutemos nuestros scripts, podremos omitirlo, en referencias a los objetos dentro de esa ventana.
nombrePropiedad
nombreMétodo([parámetros])
También funcionaría sin ningún problema, porque se asume que esas propiedades o métodos, son del objeto de mayor jerarquía (el objeto window) en el cual nos encontramos.

GESTIÓN DE VENTANAS

Un script no creará nunca la ventana principal de un navegador. Es el usuario, quien realiza esa tarea abriendo una URL en el navegador o un archivo desde el menú de

abrir. Pero sin embargo, un script que esté ejecutándose en una de las ventanas principales del navegador, si que podrá crear o abrir nuevas sub-ventanas.

El método que genera una nueva ventana es **window.open()**. Este método contiene hasta tres parámetros, que definen las características de la nueva ventana: la URL del documento a abrir, el nombre de esa ventana y su apariencia física (tamaño, color,etc.).

Por ejemplo, si consideramos la siguiente instrucción que abre una nueva ventana de un tamaño determinado y con el contenido de un documento HTML:
var subVentana=window.open("nueva.html","nueva","height=800,width=600");

Lo importante de esa instrucción, es la asignación que hemos hecho en la variable subVentana. De esta forma podremos a lo largo de nuestro código, referenciar a la nueva ventana desde el script original de la ventana principal. Por ejemplo, si quisiéramos cerrar la nueva ventana desde nuestro script, simplemente tendríamos que hacer: ***subVentana.close();***

Aquí sí que es necesario especificar subVentana, ya que si escribiéramos window.close(), self.close() o close() estaríamos intentando cerrar nuestra propia ventana (previa confirmación), pero no la subVentana que creamos anteriormente.

El objeto window representa una ventana abierta en un navegador. Si un documento contiene marcos (<frame> o <iframe>), el navegador crea un objeto window para el documento HTML, y un objeto window adicional para cada marco.

Ver anexo sobre objeto window para más información sobre propiedades y métodos

Cómo se ve en el gráfico de la jerarquía de objetos, debajo del objeto window tenemos otros objetos como el **navigator, screen, history, location** y el objeto **document**. Este objeto document será el que contendrá toda la jerarquía de objetos, que tengamos dentro de nuestra página HTML, constituyendo lo que se denomina el **modelo de objetos del documentos (DOM)** que es la parte más importante del trabajo de JavaScript en la creación de aplicaciones web.

A continuación, vamos a conocer alguno de los objetos importantes del navegador y veremos una gran lista de métodos y propiedades de todos ellos. No se trata de aprenderse esta lista, pero sí de intuir qué ayuda pueden proporcionar estos objetos a nuestras aplicaciones. Después veremos el DOM.

5.1.2 OBJETO LOCATION

Se trata de otro objeto interesante que posee el objeto window. La utilidad de location reside en el hecho de que representa la dirección URL de la propia aplicación. Esta misma propiedad es accesible mediante document.location, que es una referencia al mismo objeto.

La propiedad más importante del objeto es href que contiene la URL completa de la página. Ejemplo:

consolé.log(location.href);

Podría devolver algo como: https://www.google.es/
Más interesante aún es el hecho de que podemos ir a otra página simplemente cambiando esta propiedad:

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

Si ejecutamos este comando desde la consola, veremos cómo nos trasladamos a la página de GitHub, independientemente de la página que estuviéramos viendo. Hay una forma más cómoda incluso de obtener el mismo resultado, que es modificar directamente la propiedad location:

location ="https://github.com";

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

Propiedades del objeto Location	
Propiedad	**Descripción**
`hash`	Cadena que contiene el nombre del enlace, dentro de la URL.
`host`	Cadena que contiene el nombre del servidor y el número del puerto, dentro de la URL.
`hostname`	Cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL.
`href`	Cadena que contiene la URL completa.
`pathname`	Cadena que contiene el camino al recurso, dentro de la URL.
`port`	Cadena que contiene el número de puerto del servidor, dentro de la URL.
`protocol`	Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.
`search`	Cadena que contiene la información pasada en una llamada a un script, dentro de la URL.

Métodos del objeto Location	
Método	**Descripción**
`assign()`	Carga un nuevo documento.
`reload()`	Vuelve a cargar la URL especificada en la propiedad `href` del objeto `location`.
`replace()`	Reemplaza el historial actual mientras carga la URL especificada en cadenaURL.

5.1.3 OBJETO NAVIGATOR

Es un objeto que representa al navegador del usuario que está utilizando la aplicación web. Posee numerosas propiedades de lectura que nos permiten obtener información interesante. Quizá la más utilizada es **userAgent** que permite obtener la cadena de información del navegador mediante la que podremos saber el navegador del usuario,

su versión, motor, etc. Abriendo el navegador y escribiendo este comando en la consola, obtendremos información que varía en cada navegador:

console.log(navigator.userAgent);

Propiedades del objeto Navigator	
Propiedad	**Descripción**
`appCodeName`	Cadena que contiene el nombre en código del navegador.
`appName`	Cadena que contiene el nombre del cliente.
`appVersion`	Cadena que contiene información sobre la versión del cliente.
`cookieEnabled`	Determina si las cookies están o no habilitadas en el navegador.
`platform`	Cadena con la plataforma sobre la que se está ejecutando el programa cliente.
`userAgent`	Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades `appCodeName` y `appVersion`.

Métodos del objeto Navigator	
Método	**Descripción**
javaEnabled()	Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.

5.1.4 OBJETO SCREEN

Se trata de otra propiedad miembro de window. En este caso, esta propiedad nos permite acceder a un objeto que representa a la pantalla. Permite obtener propiedades relacionadas con la pantalla del dispositivo en el que está navegando el usuario. Propiedades interesantes de este objeto son:

PROPIEDAD O MÉTODO	USO
availTop	Primera coordenada superior de la pantalla que se puede utilizar en nuestras aplicaciones.
availLeft	Primera coordenada izquierda de la pantalla que se puede utilizar en nuestras aplicaciones.
availHeight	Número que indica el tamaño en píxeles de la altura de pantalla del usuario.
availWidth	Número que indica el tamaño en píxeles de la anchura de la pantalla del usuario.
height	Altura completa de la pantalla del usuario en píxeles.
width	Anchura completa de la pantalla del usuario en píxeles.

colorDepth	Profundidad en bits de los colores de la pantalla. El número de colores disponibles en la pantalla será 2 elevado a la cantidad devuelta por esta propiedad, por ejemplo: 2^{24}
orientation	Devuelve un objeto de tipo ScreenOrientation que sirva para saber la orientación de la pantalla. Ejemplo de uso: consolé.log(screen.orientation.type); En una pantalla normal de escritorio retornaría: landscape-primary

5.1.5 OBJETO HISTORY

Se trata del objeto que representa el historial de páginas visitadas por el usuario. El método más interesante de este objeto 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() 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))***
Este código nos permite volver a la primera página desde el historial.

5.1.6 OTRAS PROPIEDADES Y MÉTODOS DE WINDOW

El objeto window tiene otras propiedades y métodos que pueden ayudarnos a realizar algunas acciones desde JavaScript. Hay otros objetos interesantes. A continuación, se detallan algunas de esas propiedades.

PROPIEDAD O MÉTODO	USO
innerWidth	Anchura interior de la ventana. Es la anchura real disponible en el área de contenido de la ventana teniendo en cuenta su tamaño actual.
innerHeight	Altura interior de la ventana. Es la altura real disponible en el área de contenido de la ventana teniendo en cuenta su tamaño actual.
outerWidth	Anchura exterior de la ventana.
innerHeight	Altura exterior de la ventana.
screenX	Distancia de la ventana al borde izquierdo de la pantalla.
screenLeft	Idéntica a la anterior.
screenY	Distancia de la ventana al borde superior de la pantalla.
screenTop	Idéntica a la anterior.
scrollX	Índica el desplazamiento horizontal que el usuario ha realizado usando las barras de desplazamiento horizontales o arrastrando un dispositivo táctil en horizontal.
scrollY	Indica el desplazamiento vertical que el usuario ha realizado usando las barras de desplazamiento verticales o arrastrando un dispositivo táctil en vertical.
status	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.
console	Consola del navegador. Accede a todos los métodos de la

	consola, incluido console.log.
event	Objeto del evento actualmente en ejecución.
fullScreen	Booleano que indica si la aplicación se muestra a pantalla completa. Es posible que No funcione en versiones actuales Google Chrome.
localStorage	Referencia al objeto de tipo Storage que permite almacenamiento de datos en modo local sin tiempo de expiración.
getSelection()	Obtiene un objeto de tipo Selection que nos sirve para poder obtener información sobre el texto seleccionado. Si queremos, simplemente, obtener el texto seleccionado actualmente por consola, sería: *console.log(getSelection().toString())*
scroll(x,y) o bien **scroll(objetoScroll)**	Sirve para desplazar la página a una posición concreta. Permite indicar el desplazamiento mediante dos coordenadas (x horizontal e y en vertical). Ejemplo: *scroll(0,100)* Desplaza la página haciendo que la barra vertical se desplace a la posición marcada por 100 píxeles. En horizontal se desplaza totalmente a la izquierda. Se puede pasar, en lugar de las dos coordenadas, un objeto de tipo **ScrollToOptions** (algunos navegadores como Edge o Explorer no aceptan este parámetro todavía). Los objetos ScrollToOptions tienen estas tres propiedades: ■ top. Desplazamiento vertical. ■ left: Desplazamiento horizontal. ■ behavior: Comportamiento. Por ahora admite como valores: auto (se desplaza sin animación) y smooth (desplazamiento con animación suave). Ejemplo: scroll({ top:1000, left:0, behavior:"smooth" El documento se desplaza automáticamente a la posición 1000 píxeles de forma progresiva con una animación.
scrollBy(x,y) o bien **scrollBy(objetoScroll)**	Los parámetros funcionan igual que con la propiedad anterior, pero ahora el desplazamiento es relativo respecto 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
scrollTo(x,y) o bien **scrollTo(objetoScroll)**	Los parámetros funcionan igual que con la propiedad scroll. La diferencia es que los valores indicados son coordenadas y no píxeles de desplazamiento (normalmente ambas cosas coinciden)
stop()	Detiene la carga de la página.
find(texto)	Busca el texto en el documento actual y devuelve true o false si lo encuentra. Además, la mayoría de navegadores resaltan la palabra dentro de la página a la vez que realizan un scroll hacia ella.
getComputedStyle(elemento)	Permite obtener la información de las propiedades CSS en

	uso por parte de un elemento.
open(URL,nombreVentana, ajustes)	Abre una ventana y muestra el contenido de la URL indicada. Opcionalmente, se puede indicar un nombre para la ventana y los ajustes de la misma (tamaño, barras de desplazamiento, etc.).

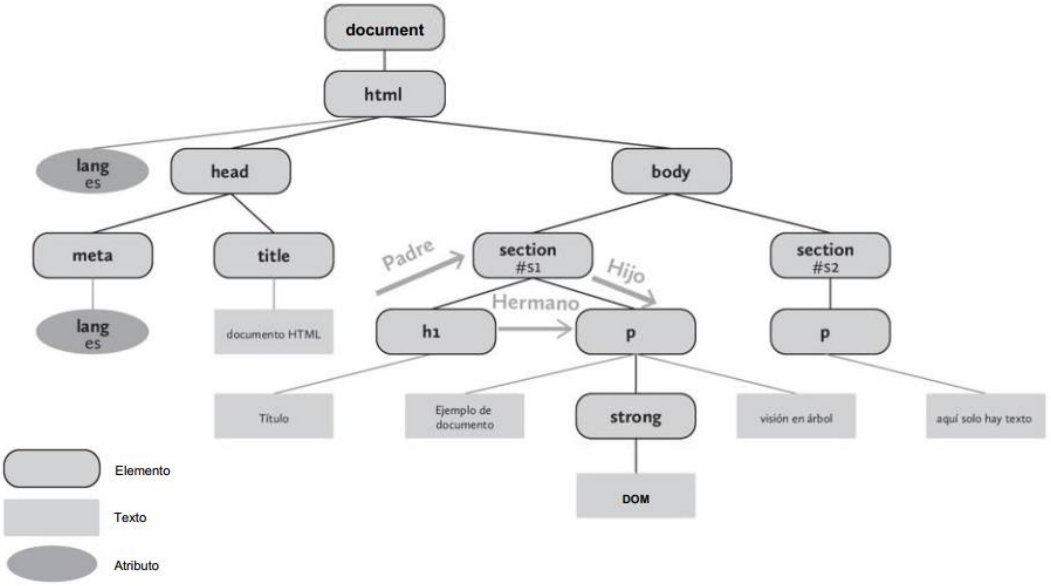
5.2 DOM. MODELO DE OBJETOS DEL DOCUMENTO

JavaScript se ideó para que fuera un lenguaje con capacidad de manipular el contenido y apariencia de una página web de forma dinámica. Para hacer esto posible, los elementos de la página web se tienen que poder manipular como una serie de objetos relacionados. Así apareció el término **DOM**, **Document Objects Model (Modelo de Objetos del Documento)** que define una página web como una estructura organizada de objetos que forman un árbol de recorrido.

Si, por ejemplo, tenemos este código HTML:

```
<!DOCTYPE html>
<html lang=”es">
<head>
      <meta charset="UTF-8">
      <title>Documento HTML</title>
</head>
<body>
   <section id="sl">
         <hl>Título</hl>
         <p>Ejemplo de documento <strong>DOM</strong>Vísión en árbol</p>
   </section>
   <section id="s2">
         <p>Aquí sólo hay texto</p>
   </section>
</body>
</html>
```

El modelo de objetos del documento (DOM) define un tipo de estructura de objetos en forma de árbol donde los nodos representan objetos del documento y cada nodo posee un nodo padre a través del cual nos acercamos a la raíz del árbol. Esta estructura manipulativa, no solo se ideó para documentos HTML, sino que funciona de la misma manera para manipular documentos XML.
Para el documento anterior tendríamos la siguiente estructura:



5.2.1 EL OBJETO DOCUMENT

La forma de poder acceder a los objetos del documento en javaScript es a través de un objeto llamado precisamente **document**. Este objeto en realidad es una propiedad del objeto window. Podemos indicar simplemente la palabra document para acceder a este objeto, 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.

TIPOS DE NODOS

Uno de los métodos más habitual para acceder a los elementos de un documento es **getElementById** del objeto document. Con el siguiente código podríamos acceder a la primera sección del documento anterior:
let seccion=document.getElementById("s1");

Este código hace que la variable sección sea una referencia al elemento identificado como s1. Es decir, seccion **es un nodo del documento**. Todos los nodos poseen una propiedad llamada **nodeType**. Si la mostramos en este caso:
console.log(sección.nodeType); // visualiza 9

La lista de posibles valores de nodeType es la siguiente:

VALOR	TIPO DE NODO	CONSTANTE RELACIONADA
1	Elemento	**Node. ELEMENT.NODE**
2	Atributo	**Node.ATTRIBUTE_NODE**
3	Texto	**Node.TEXT_NODE**
4	Apartado CDATA	**Node.CDATA_SECTION_NODE**
5	Referencia a entidad	**Node.ENTITY_REFERENCE_NODE**
6	Entidad	**Node.ENTITY_NODE**
7	Instrucción de procesado	**Node.PROCESSING_INSTRUCTION_NODE**
8	Comentario	**Node.COMENT_NODE**
9	Documento completo	**Node.DOCUMENT_NODE**
10	Nodo de tipo de documento	**Node.DOCUMENT_TYPE_NODE**
11	Nodo de fragmento de código	**Node.DOCUMENT_FRAGMENT_NODE**
12	Nodo de anotación	**Node.NOTATION_NODE**

Algunas entradas están muy relacionadas con documentos XML por lo que no nos serán útiles para manipular documentos HTML. Normalmente, nos bastará con las tres primeras entradas de esta tabla. Las constantes definidas en el objeto Node, nos permiten no tener que aprender los números y manejar en su lugar las constantes que son más entendibles.

La propiedad **nodeName**: nos permite obtener el nombre del nodo.
console.log(s1.nodeName); // escribe SECTION
console.log(document.nodeName); // escribe #document

La lista de posibles valores de nodeName es la siguiente:

TIPO DE NODO	VALOR DEVUELTO POR NODENAME
Elemento	Nombre de la etiqueta del elemento
Atributo	Nombre del atributo
Texto	#text
Comentario	#comment
Documento	#document

Finalmente, la propiedad **nodeValue** es capaz de devolver el valor del nodo, aunque en el caso del documento completo nodeValue devuelve null.
Estas tres propiedades (**nodeType, nodeName y nodeValue**) facilitan obtener la información más importante sobre un nodo a la hora de manipularlo.

5.3 SELECCIÓN DE ELEMENTOS DEL DOM

EL objeto document proporciona numerosos métodos para seleccionar elementos. Las últimas revisiones del DOM han añadido nuevos métodos que facilitan mucho la labor del desarrollador.

5.3.1 SELECCIÓN POR EL IDENTIFICADOR

Se trata de seleccionar un nodo del documento por el valor del atributo id. El método que lo permite es ***getElementById*** que recibe un string con el valor buscado. Retorna como resultado el elemento en sí o bien el valor null si ese identificador no se encuentra en el elemento.
let s1=document.getElementById("s1");

5.3.2 SELECCIÓN POR ETIQUETA

La selección por etiqueta permite seleccionar todos los elementos que tengan ese nombre de etiqueta. El método que lo consigue es **getElementsByTagName**. En este caso se devuelve una lista de elementos de tipo **NodeList**. Es decir, no se selecciona un elemento (como ocurría con getElementById), sino todos los elementos que usen esa etiqueta.
let elementosP = document.getElementsByTagName("p"); // elementos es una referencia a todos los elementos de tipo p.

console.log(elementosP.length); // escribe 2
console.log(elementosP[0]); // escribe el contenido del primer elemento p

El método getElementsByTagTagName lo tienen todos los elementos. Si en lugar de utilizar document, usamos un elemento concreto, entonces se buscan los elementos con esa etiqueta dentro del que digamos. Ejemplo:

```
<main id="principal">
        <p>Uno</p>
        <p>Dos</p>
        <p>Tres</p>
</main>
<p>Cuatro</p>
<p>Cinco</p>
<script>
                    let principal=document.getElementBy!d("principal");
    let interiores=principal.getElementsByTagName("p");
    consolé.log(interiores.length)
</script>
```

5.3.3 OBJETOS NODELIST

El selector por etiqueta nos ha permitido conocer a los objetos **NodeList** que representan una colección de nodos del DOM, es decir, una serie de elementos del documento.

Aunque los ***objetos NodeList*** no son realmente arrays, se manejan en gran medida como si lo fueran. ***Se pueden usar todos los bucles de recorrido vistos con los arrays, se puede acceder a cada elemento indicando su índice entre corchetes y dispone de propiedades comunes a los arrays como length o forEach***. Sin embargo, otros métodos de los arrays como slice, sort o join, no están disponibles.

Si deseamos disponer de esos métodos y de otros, tendremos que convertir este tipo de objetos en un array (lo cierto es que suele ser necesaria esta conversión), lo cual es posible mediante el operador de propagación:
let arrayElementosP = [...elementoP];

O, también:

let arrayElementosP = Array.from(elementosP);

5.3.4 SELECCIÓN POR CLASE

Se trata de seleccionar todos los elementos que tienen asignada una clase determinada de CSS.
Por ejemplo:

```
<ul>
    <li class="verdura">Cebolla</li>
    <1 i c1ass="verdura">Ajo</1 i >
    <li class="carne">Pollo</li>
    <li class="verdura">Lechuga</li>
    <1 i class="carne">Ternera</li>
    <li class="hidrato">Pan</I i>
    <1 i class="hidrato">Arroz</li>
    <1 i class="carne">Cerdo</li>
    <1 i class="verdura">Tomate</li>
    <1 i class="hidrato">Lentejas</li>
</ul>
```

En esta lista de alimentos cada elemento se relaciona con una clase concreta, eso permite aplicar un CSS distinto a cada clase de alimento. Si deseamos operar desde JavaScript con los elementos de una clase concreta, disponemos del método: getElementsByClassName , el cual nos devuelve un objeto de tipo NodeList con todos los elementos que sean de la clase indicada:
let verduras=document.getElementsByClassName("verdura");

5.3.5 SELECCIÓN POR SELECTOR CSS

Indudablemente el selector más poderoso es ***querySelectorAll***. Permite indicar, como texto, cualquier selector válido CSS. Si la sintaxis del selector no es correcta, entonces devolverá un error. Lo que devuelve, nuevamente, es una lista de elementos con aquellos que cumplan el selector.
let verduras=document.querySelectorAll(".verdura"); // Selecciona todos los elementos de la clase verdura

//Selecciona los elementos li que están dentro de ul
document.querySelectorAll("ul li");

//Selecciona el primer elemento li de cada lista
document.querySelectorAll("li:first-of-type");

//Selecciona el primer elemento li de cada lista si es de clase verdura
document.querySelectorAll("li.verdura:first-of-type")

Hay un método llamado ***querySelector*** a secas (sin la palabra all) que funciona igual, pero no devuelve una lista, sino el primer elemento en el documento que cumpla el criterio. No suele ser muy habitual su uso.

5.4 OBTENER Y MODIFICAR DOM

5.4.1 MANIPULACIÓN DE ATRIBUTOS

OBTENER EL VALOR DE UN ATRIBUTO
Los elementos del DOM disponen del método **getAttribute** para obtener el valor de un atributo concreto. Por ejemplo:

```
let lis=document.querySelectorAll("li");
for (let li of lis){
 console.log(li.getAttribute("class"));
}
```

Este código recorre cada elemento de tipo li y nos muestra por consola la clase CSS de cada elemento. Si un elemento no tiene el atributo indicado, getAttribute devuelve null.

MODIFICAR EL VALOR DE UN ATRIBUTO
El método **setAttribute** es el que permite esta operación. El primer parámetro será el nombre del elemento y el segundo el nuevo valor.
elemento.setAttribute("class","verdura");

ELIMINAR UN ATRIBUTO

El método **removeAttribute** seguido de un nombre de atributo, nos permite eliminar el atributo que tenga ese nombre del elemento.

AÑADIR Y QUITAR ATRIBUTO DE FORMA RÁPIDA

El método **toggleAttribute** permite añadir al elemento el atributo que se indique si no está añadido ya, o quitar el atributo con ese nombre si ya está añadido. Si la primera vez que usamos el método toggleAtribute, añade el atributo, la siguiente lo eliminará, la siguiente lo volverá a añadir y así sucesivamente.

SABER SI UN ELEMENTO TIENE ATRIBUTOS

El método **hasAttribute** devuelve verdadero si el elemento tiene el atributo cuyo nombre se indica y falso si no los tiene.

OBTENER TODOS LOS ATRIBUTOS DE UN ELEMENTO

La propiedad atributes de un elemento permite obtener todos los atributos de un elemento. Lo que **devuelve es un objeto de tipo NamedNodeList** que no es una estructura de array por lo que no tenemos acceso a la mayoría de propiedades del array. Pero sí podemos acceder por número de índice y usar la propiedad length.

Cada elemento de la lista que devuelve esta propiedad posee una propiedad llamada **name** que contiene el nombre del atributo y **value** que contiene su valor. Por ejemplo, si tenemos este código HTML:

```
<form>
    <label for="edad">Escriba su edad></label>
    <input type="number" id="edad" name="edad" min="18" max="65">
</form>
```

Para navegar por los atributos del cuadro numérico y mostrarlos por consola podríamos escribir este código:

```
let listaAtributos = document.getElementById("edad").attributes;
for(let atributo of listaAtributos){
 console.log(`Atributo: ${atributo.name} '+ "Valor: ${atributo.valué}`);
```

5.4.2 MANIPULACIÓN DEL CONTENIDO DE LOS ELEMENTOS

PROPIEDAD TEXTCONTENT

La propiedad **textContent** de un elemento permite obtener y modificar el texto que contiene ese elemento. Hay que tener en cuenta que un elemento puede tener dentro más elementos y que esta propiedad obtendrá el texto de todos ellos. Por ejemplo, si observamos este código:

```
<main>
   <hl>Sistema Solarc/hl>
   <h2>Sol</h2>
   <p id="pSol">
      El Sol, una estrella de tipo espectral <strong>G2</strong> que
      contiene más del 99,85 % de la masa del sistema. Con un diámetro
      de 1 400 000 km, se compone de un 75 % de hidrógeno, un 20 % de
      helio y 5 % de oxígeno, carbono, hierro y otros elementos.
   </p>
   <h2>Planetas</h2>
   <p id="pPlanetas">
      Los planetas, divididos en planetas interiores (también llamados
      terrestres o telúricos) y planetas exteriores o gigantes. Entre
      estos últimos <strong>Júpiter</strong> y <strong>Saturno</strong>
      se denominan gigantes gaseosos, mientras que Urano y Neptuno
      suelen nombrarse gigantes helados. Todos los planetas gigantes
      tienen a su alrededor anillos.
   </p>
</main>
```

Al ejecutar esta instrucción:
console.log(document.getElementById("pSol").textContent);

Escribe el texto del párrafo indicado. Sin embargo, dentro de ese elemento hay otro de tipo strong y su texto aparece también en el resultado a la vez que la propia etiqueta strong desaparece del resultado. Es decir, solo recoge el texto interior (todo el texto) y no las etiquetas.

A través de esta propiedad podemos cambiar el contenido:
document.getElementById("pSol").textContent = "Soy el Sol";

Si observamos el resultado en la página (ese código lo podemos escribir en la consola del navegador), veremos que aunque hemos indicado que la palabra Sol se encuentre dentro de etiquetas strong, estas no funcionan, se toman de forma literal. La razón es que textContent tiene capacidad para modificar el texto de un elemento, pero no los elementos interiores.

PROPIEDAD INNERHTML

Se trata de una propiedad similar a la anterior, pero esta sí lee y manipula las etiquetas HTML. Por ejemplo:
console.log(document.getElementById("pSol").innerHTML);

Este código muestra:

```
El Sol, una estrella de tipo espectral <strong>G2</strong> que contiene
más del 99,85 % de la masa del sistema. Con un diámetro de 1 400 000 km,
se compone de un 75 % de hidrógeno, un 20 % de helio y 5 % de oxígeno,
carbono, hierro y otros elementos.
```

Parece el mismo resultado que el obtuvimos con textContent. Pero la palabra G2 está contenida en una etiqueta strong que con textContent no podíamos ver. ***La diferencia es que innerHTML sí lee las etiquetas***.

Y no solo las lee, ***puede modificar el contenido usando etiquetas HTML***:
document.getElementById("pSol").innerHTML = "Soy el Sol";

Si vemos el resultado, ahora sí veremos que entiende las etiquetas y realmente vemos en negrita la palabra Sol.