Desarrollo web en entorno cliente

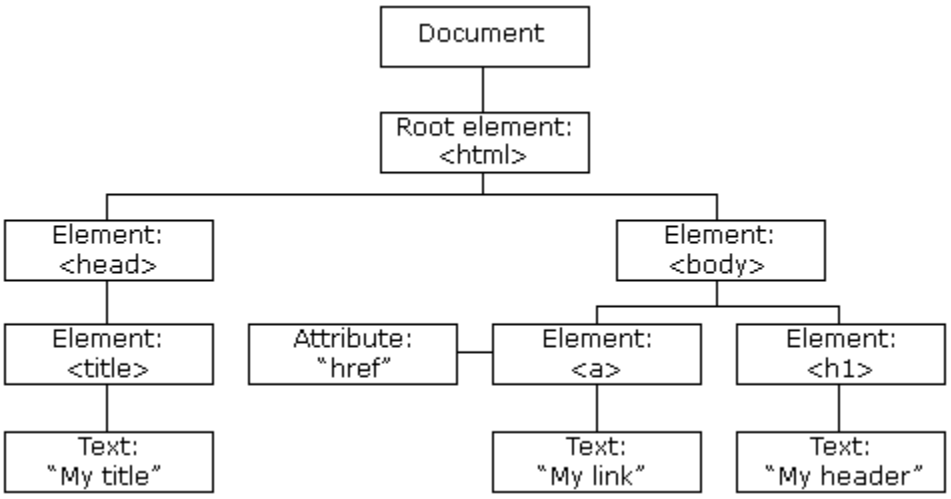
Tema 5: DOM / BOM

1. DOM (Document Object Model)

Aunque es posible ejecutar código JavaScript del lado del servidor, éste surgió para dotar de funcionalidad e interactividad al lado del cliente. DOM (Document Object Model) y BOM (Browser Object Model) ofrecen objetos que permiten interactuar con una página web (DOM) y con el navegador donde ésta se abra (BOM).

DOM permite identificar cada elemento de una página para poder manipularlo, modificar sus propiedades, e incluso mostrarlo o no dependiendo del momento y situación.

El W3C definió el estándar DOM, que define qué elementos se considera que conforman una página web, cómo se nombran, cómo se relacionan entre sí, cómo se puede acceder a ellos, etc. Cuando se carga una página web, el navegador crea un modelo de objetos de la misma. Este modelo se construye como un árbol, donde la raíz del árbol es **document** y de este nodo cuelgan el resto de elementos HTML. Cada uno constituye su propio nodo y tiene subnodos con sus *atributos*, *estilos* y elementos HTML que contiene:



Cada etiqueta HTML suele originar 2 nodos:

- element: correspondiente a la etiqueta
- text: correspondiente a su contenido (lo que hay entre la etiqueta de apertura y la de cierre)

Realmente, si el árbol de ejemplo estuviese completo, contendría también nodos los saltos de línea, tabuladores, espacios, comentarios, etc.

Cada nodo es un objeto con sus propiedades y métodos. Los métodos de DOM son acciones que se pueden ejecutar sobre elementos HTML, mientras que las propiedades son valores de los mismos que se puede establecer o eliminar.

1.1. Acceso a nodos

Acceso directo a nodos

Entre otros, los métodos más utilizados para acceder a los nodos son:

- **.getElementById(id)**: devuelve el nodo cuyo id coincida con el valor pasado como parámetro:

```
let nodo = document.getElementById('encabezadoPrincipal');
// nodo contendrá el nodo cuyo id sea encabezadoPrincipal
```

- **.getElementsByClassName(clase)**: devuelve una colección, que contendrá todos los nodos cuya clase coincida con la indicada. Una colección es similar a un array, ya que se accede a ella mediante un índice, pero no puede aplicársele los métodos propios de arrays como filter, map, etc. Habría que convertirla primero a array:

```
let nodos = document.getElementsByClassName('encabezadosSec');
// nodos contendrá todos los nodos cuya clase es encabezadosSec
```

- **.getElementsByTagName(etiqueta)**: devuelve una colección con todos los nodos de la etiqueta HTML indicada:

```
let nodos =document.getElementsByTagName('a');
// nodos contendrá todos los nodos de tipo <a>
```

- **.querySelector(selector)**: devuelve el primer nodo seleccionado por el selector CSS indicado:

```
let nodo = document.querySelector('a.encabezadosSec');
// nodo contendrá el primer hipervínculo de clase encabezadosSec
```

- **.querySelectorAll(selector)**: devuelve una colección con todos los nodos seleccionados por el selector CSS indicado:

```
let nodos = document.querySelectorAll('a.encabezadosSec');
// nodos contendrá todos los hipervínculos de clase encabezadosSe
```

Cuando se hace uso de estos métodos, si se ejecutan sobre el nodo **document** se seleccionará sobre la página completa, pero es posible aplicarlos sobre un nodo de diferente tipo y entonces la búsqueda se realizaría entre los descendientes de dicho nodo.

Además, existe una serie de atajos que permiten obtener de forma rápida algunos elementos comunes:

- **document.documentElement**: devuelve el nodo del elemento <html>.
- **document.head**: devuelve el nodo del elemento <head>.
- **document.body**: devuelve el nodo del elemento <body>.
- **document.title**: devuelve el nodo del elemento <title>.
- **document.link**: devuelve una colección con todos los hiperenlaces del documento.
- **document.anchor**: devuelve una colección con todas las anclas(enlaces) del documento.
- **document.form**: devuelve una colección con todos los formularios del documento.
- **document.images**: devuelve una colección con todas las imágenes del documento.
- **document.scripts**: colección con todos los scripts del documento.
- **elemento.nombreElementoHijo**: devuelve el hijo que tenga como valor del atributo name “nombreElementoHijo”. También es posible usarlo con el id.

EJERCICIO 1

Descarga el proyecto "TrabajarConDom" que encontrarás en el Aula Virtual y abre la página en un navegador. Realiza las siguientes operaciones a través de la consola (Ve guardando todo lo que hagas en un archivo):

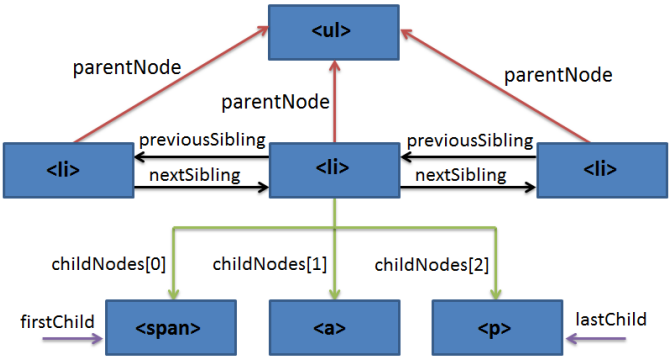
- Recupera el elemento con id "apellido1".
- Recupera todos los párrafos de una vez.
- Recupera, de una sola vez, todos los párrafos del div con id "seccionTercera".
- Recupera todos los elementos de tipo input.
- Recupera los elementos de tipo input con nombre "sexo".
- Recupera los elementos de la lista de la clase "par".

Acceso a nodos a través de otros

Si la referencia que se tiene para acceder a un nodo es otro nodo, se puede acceder al primero teniendo en cuenta la relación que tiene con el segundo:

- **elemento2.parentElement**: devuelve el elemento padre de elemento2.
- **elemento2.children**: devuelve una colección con todos los elementos HTML que sean hijos de elemento2.
- **elemento2.childNodes**: devuelve una colección con todos los hijos de elemento2, incluyendo comentarios y nodos de tipo texto.
- **elemento2.firstElementChild**: devuelve el elemento HTML que es el primer hijo de elemento2.
- **elemento2.firstChild**: devuelve el nodo primer hijo de elemento2, teniendo en cuenta nodos de tipo texto y comentarios.
- **elemento2.lastElementChild**, **elemento2.lastChild**: igual que los anteriores, pero con el último hijo.
- **elemento2.nextElementSibling**: devuelve el elemento HTML que es el siguiente hermano de elemento2.
- **elemento2.nextSibling**: devuelve el nodo que es el siguiente hermano de elemento2, teniendo en cuenta nodos de tipo texto o comentarios.
- **elemento2.previousElementSibling**, **elemento2.previousSibling**: igual que los anteriores, pero con el hermano anterior.
- **elemento2.hasChildNodes**: indica si elemento2 tiene o no nodos hijos.
- **elemento2.childElementCount**: devuelve el nº de nodos hijo de elemento2.

De estas propiedades lo recomendable es usar las que solo devuelven elementos HTML, a no ser que por alguna razón concreta interese recuperar comentarios, etc.



EJERCICIO 2

Siguiendo con el proyecto "TrabajarConDom", abre la página en un navegador y realiza las siguientes operaciones a través de la consola del mismo, documentando cada uno de los puntos (Busca dos soluciones diferentes para ejecutar cada punto):

- Recupera el primer párrafo que hay dentro del div "seccionPrimera".
- Recupera el tercer párrafo del div anterior.
- El último elemento de la lista.
- La label del input nombre.

1.2. Propiedades de un nodo

A continuación, se van a ver las principales propiedades que tiene cualquier nodo:

- **elemento.innerHTML**: Devolverá todo lo que se encuentra entre las etiquetas de apertura y cierre de un elemento HTML, incluido otros elementos HTML.
- **elemento.textContent**: Devolverá el contenido textual que se encuentre entre las etiquetas de apertura y cierre de un elemento HTML. En este caso ignora otros elementos HTML.
- **elemento.value**: Esta propiedad devuelve la propiedad "value" de un elemento <input> o, en el caso de un <input> de tipo text, lo que hay escrito en él. En este tipo de elementos no se puede utilizar las propiedades anteriores ya que son elementos sin etiqueta de cierre.

Otras propiedades o métodos, algo secundarias pero igualmente utilizadas son:

- **elemento.innerText**: Mismo comportamiento que textContent.
- **elemento.focus() / elemento.blur()**: Propiedades para dar/quitar el foco a un elemento (input, etc).
- **elemento.clientHeight /elemento.clientWidth**: devuelve el alto/ancho visible del elemento.
- **elemento.offsetHeight /elemento.offsetWidth**: devuelve el alto / ancho total del elemento.
- **elemento.clientLeft /elemento.clientTop**: devuelve la distancia de elemento al borde izquierdo / superior.
- **elemento.offsetLeft / elemento.offsetTop**: devuelve los píxels que se ha desplazado elemento a la izquierda / abajo.

EJERCICIO 3

- Continuando con el ejercicio2:
  - Recupera el innerHTML del elemento ul y su textContent.
  - Compara los resultados de las dos recuperaciones del punto anterior.
  - Recupera el valor del primero input radio de sexo.
  - Busca como recuperar el valor del sexo que está seleccionado.
- Crea el archivo trabajarConDom.js y programa la siguiente funcionalidad:
  - Al pulsar el botón Aceptar, se dará el foco a la caja del nombre.
  - Al pulsar el botón Cancelar, el foco se quitará de la caja del nombre.

1.3. Manipular el árbol DOM

DOM proporciona también métodos que permiten modificar el árbol DOM y, por lo tanto, modificar una página web. Algunos de los métodos más utilizados son:

- **document.createElement('etiqueta')**: crea un nuevo elemento HTML del tipo que se indica por parámetro. Solo l ocrea, no lo añade a la página.
- **document.createTextNode('texto')**: crea un nuevo nodo de tipo texto con el texto que se le pasa como parámetro. Un nodo de tipo texto habrá que añadirlo después a un documento HTML.
- **elemento.appendChild(nuevoNodo)**: método que añade el nodo que se le pasa como parámetro como último hijo del elemento.
- **elemento.insertBefore(nuevoNodo, nodo)**: este método permite añadir un nuevo nodo como hijo del elemento colocándolo antes del nodo hijo pasado como segundo parámetro.
- **elemento.removeChild(nodo)**: Método que elimina el nodo pasado como parámetro como hijo del elemento y que, por tanto, desaparecerá de la página.
- **elemento.replaceChild(nuevoNodo,viejoNodo)**: sustituye, como hijo de elemento, el nodo pasado como segundo parámetro por el que se pasa como primero.
- **elementoAClonar.cloneNode(boolean)**: método que devolverá un clon del elemento (si el parámetro es false) o un clon del elemento con todos sus descendientes (si se le pasa true).

Si se quiere añadir un nodo ya existente como hijo de otro elemento, con appendChild se eliminará el nodo de su ubicación inicial. Si lo que se pretende es que dicho nodo esté en ambos sitios habría que clonarlo (crear uno nuevo igual al original) y añadir ese clon a la nueva posición.

EJERCICIO 4

Continúa trabajando con el proyecto TrabajarConDom, en el documento trabajandoConDom.js crea una función para cada una de las siguientes tareas:

- Crea un elemento de tipo h2 con el texto “Lorem Ipsum de nuevo”. Inclúyelo como primer elemento hijo del div seccionTercera cuando se pulse sobre el primer párrafo de dicha sección.
- El tercer párrafo del div seccionPrimera pasará a ser el primero del div seccionTercera (sin eliminar ninguno de los que ya tiene esta última). Esta acción se desencadenará cuando se pulse en ese tercer párrafo.
- Crea un div nuevo que se añadirá tras el div seccionSegunda(una vez que este se haya cerrado). Clona (con todos sus descendientes) el elemento div que contiene la lista y añádelo como elemento hijo del nuevo div creado. Esta acción se desencadenará cuando se haga click sobre el encabezado “Frase inicial párrafos Lorem Ipsum”.
- Elimina la última fila de la lista nueva. Esta acción se desencadenará al pasar el ratón por encima del encabezado h1 que hay en la página.

Modificar el DOM con ChildNode

ChildNode es una interfaz que permite manipular del DOM de forma más sencilla pero no está soportada en los navegadores Safari de IOS. Incluye los métodos:

- **elemento.before(nuevoNodo)**: añade el nodo *nuevoNodo* antes del nodo elemento.
- **elemento.after(nuevoNodo)**: añade el nodo pasado como parámetro después del nodo elemento.
- **elemento.replaceWith(nuevoNodo)**: sustituye el nodo elemento por el nodo *nuevoNodo* pasado como parámetro.
- **elemento.remove()**: elimina el nodo elemento.

EJERCICIO 5

- Haz una nueva copia del proyecto TrabajarConDom tal y como ha quedado tras terminar el ejercicio 3. En dicha copia incluye el código necesario para realizar las mismas tareas que en el ejercicio 4 pero utilizando ChildNode.
- Compara los resultados de este ejercicio con los obtenidos en el ejercicio 4 y constata si hay alguna diferencia.

1.4. Atributos de los nodos

Otra opción que DOM ofrece es recuperar y modificar los valores de los atributos de un elemento HTML, así como eliminar atributos o crear nuevos y asignarlos al elemento. Algunos de las propiedades y métodos utilizados para ello son:

- **elemento.attributes**: devuelve un array con todos los atributos de elemento en cuestión.
- **elemento.hasAttribute('nombreAtributo')**: método que devolverá true, si el elemento tiene definido el atributo pasado como parámetro, o false en caso contrario.
- **elemento.getAttribute('nombreAtributo')**: este método, por su parte, devolverá el valor que tengas asignado el atributo pasado como parámetro. Para algunos atributos (como id, title o class) es posible recuperar su valor haciendo **elemento.nombreAtributo**:

```
let lista=document.getElementsByTagName('ol')[0];
lista.id='listaOrdenada'; //realiza la misma operación que:
lista.setAttribute('id', ' listaOrdenada ');
```

- **elemento.setAttribute('nombreAtributo', 'valor')**: el atributo *nombreAtributo* del elemento pasará a tener como valor el pasado como segundo parámetro. También es válido hacerlo con la sentencia **elemento.atributo="valor"**.
- **elemento.removeAttribute('nombreAtributo')**: opción para eliminar el atributo cuyo nombre se ha pasado como parámetro del elemento.

EJERCICIO 6

Continúa con el proyecto TrabajarConDom. Incluye el código adecuado para que:

- Cuando se pulse sobre el primer elemento h2:
  - En caso de que estén visibles los párrafos segundo y quinto de la página, desaparezcan.
  - En caso contrario, aparecerán en su posición original.
  - Se añada al valor del atributo size, de los input de tipo texto, 5.
- Cuando se pase el ratón por encima del texto "Escoge el sexo":
  - Se quite el atributo name a los radio buttons. Comprueba cómo se comportan ahora.
- Cuando pase el ratón por encima del título del formulario:
  - Se añada un nuevo atributo name para los radio button, con un valor diferente al que tuvieran de inicio.

1.5. Estilos de los nodos

En una página web dinámica es muy normal que el estilo de los elementos cambie dependiendo de la interacción del usuario con los mismos.

A los estilos de un elemento se accede a través del atributo style de este último, el cual tendrá como propiedades todos los estilos posibles. Estas propiedades se corresponderán con la propiedad CSS solo que referenciada con la sintaxis camelCase en lugar de kebab-case:

```
//Para cambiar la propiedad background-color de un elemento sería:
elementoDiv1.style.backgroundColor = "blue";
```

Aunque es posible cambiar cualquier estilo a cualquier elemento, no es tan común modificar una propiedad concreta, sino que más bien se ponen o quitan clases que harán que se le apliquen o no los estilos definidos para ellas en el CSS.

Atributos de clase

Ya que los estilos de una página no deben aplicarse directamente a los elementos en el documento HTML, sino que es conveniente configurarlos en un documento css externo, lo suyo es asignar clases a los elementos, para que se aplique el estilo definido para la clase.

Ya se ha visto en el apartado anterior, que se podría cambiar el valor del atributo class a través de una de las siguientes sentencia s:

```
elemento.setAtttribute("class", "nombreDeClase");
elemento.className = "nombreDeClase";
```

Pero como es una acción muy común, existe también la posibilidad de utilizar la propiedad classList que devuelve una colección de todas las clases que tiene el elemento:

```
<span class="negrita resaltado">
…
let clases=elemento.classList; // devuelve la colección ["negrita",
"resaltado"]
```

Los métodos que permiten trabajar con las clases asignadas a un elemento son:

- **elemento.classList.add("nombreClase")**: añade al elemento la clase pasada en caso de que no la tenga ya.
- **elemento.classList.remove("nombreClase")**: elimina del elemento la clase pasada en caso de que la tenga asignada.
- **elemento.classList.toogle("nombreClase")**: añade la clase pasada si no la tiene o la elimina en caso de que si la tenga.
- **elemento.classList.contains("nombreClase")**: devolverá un valor indicando si el elemento tiene, o no, asignada la clase pasada como parámetro.
- **elemento.classList.replace("claseVieja","claseNueva")**: reemplaza claseVieja (ya asignada) por claseNueva.

No todos los navegadores soportan classList o todas las propiedades de classList, por lo que habrá que tenerlo en cuenta a la hora de programar. Por ejemplo, si se quiere añadir una clase a las que ya tenga asignadas un elemento sin usar classList:

```
let clases = elemento.className.split(" ");
if (clases.indexOf("principal") == -1) {
    elemento.className += " " + 'principal';
}
```

EJERCICIO 7

Partiendo del proyecto resultado del ejercicio 6, incluye el código necesario para:

- Incluye una nueva regla de estilo en el documento css para el selector .parrafosNormal. Dale el color de fondo y tipo de letra que quieras.
- Incluye otra regla para el selector .parrafosEspecial. Asígnale un color de fondo diferente al anterior y la letra que sea en negrita.
- Asigna a todos los párrafos del documento el estilo establecido para .parrafosNormal.
- Utilizando DOM, cuando se haga doble click sobre el primer párrafo de la página:
  - Asigna a todos los párrafos del documento, cuya posición entre los párrafos sea múltiplo de 3(posición física en la página), el estilo establecido para la clase parrafosEspecial (eliminando cualquier otro estilo anteriormente asignado).
  - Para los elementos de la lista, cambia el color de fondo a los que tengan asignada la clase par.

2. BOM

Si DOM permite interactuar con la página web, BOM posibilita la interacción con el navegador en el que esté abierta.

Con BOM se podrá abrir, cambiar y cerrar ventanas, ejecutar código en cierto tiempo (establecer timers), obtener información del navegador, ver y modificar propiedades de la pantalla, gestionar cookies, etc.

El problema que surge al utilizar BOM es que ninguna entidad se encarga de estandarizarlo, por lo que un navegador podría comportarse de forma diferente al resto.

2.1. Timers

Para controlar los tiempos, los timers permiten ejecutar código en un futuro, indicando tiempos de espera en milisegundos. Hay 2 tipos:

- **setTimeout(función, milisegundos)**: mediante este método se puede indicar que se ejecute la función indicada, una sola vez, cuando pasen los milisegundos que establezca el segundo parámetro.
- **setInterval(función, milisegundos)**: este método por el contrario ejecuta, cada vez que transcurran los milisegundos, la función indicada hasta que se cancele el timer.

A ambos métodos se les pueden pasar más parámetros tras los milisegundos, y serán los parámetros que recibirán como entrada las funciones a ejecutar. Además, en los dos casos, el valor devuelto será un identificador que permite cancelar la ejecución mediante los métodos:

- clearTimeout(identificador)
- clearInterval(identificador)

En el siguiente ejemplo se ve la aplicación de ambos timers:

```
let idTimeout=setTimeout(() => {
        alert('Tiemout que se ejecuta, una vez, pasados 3
segundos');
    }, 3000);


let i=1;
let idInterval=setInterval(() => {
        alert('Interval cada 3 seg. Ejecución nº: '+ i++);
        if (i==5) {
            clearInterval(idInterval);
            alert('La función se ha ejecutado 5 veces');
        }
    }, 3000);
```

EJERCICIO 8

- Prueba en la consola del navegador los ejemplos anteriores.
- Crea una página con un botón que:
  - 5 segundos después de cargarse mostrará un mensaje de bienvenida al usuario.
  - 7 segundos después de cargarse mostrará de nuevo el mensaje de bienvenida. Crea una sola función que muestre el mensaje, con el método correspondiente se llamará a esa función las dos veces.
  - Cada vez que se pulse el botón:
    - Si no se está ejecutando nada, se comenzará a mostrar por consola, cada 3 segundos, el mensaje "Han pasado X segundos", incrementándose X cada vez en 3 segundos.
    - Si está en ejecución la acción descrita en el punto anterior, se le pondrá fin.

2.2. Objeto Window

Objeto que representa una ventana abierta en el navegador. Al tratarse del objeto principal puede omitirse la referencia al mismo cuando se llama a sus propiedades o métodos. Por ejemplo, al utilizar alert() (realmente es window.alert()), prompt()(window.prompt()), los métodos de timers visto en el punto anterior(window.setTimeout() o window.setInterval()), la misma propiedad document (window.document.getElementById()), etc.

Algunas de sus propiedades y métodos son:

- **.name**: nombre de la ventana actual.
- **.screenX/.screenY**: distancia de la ventana a la esquina izquierda/superior de la pantalla.
- **.outerWidth/.outerHeight**: ancho/alto total de la ventana, incluyendo la toolbar y la scrollbar.
- **.innerWidth/.innerHeight**: medidas del documento, sin toolbar ni scrollbar.
- **.open(url, nombre, opciones)**: abre una nueva ventana. Devuelve el nuevo objeto Window. Algunas de las opciones que se pueden especificar son:
  - url: url de la página que se va a mostrar.
  - name: atributo target o nombre para la ventana.
  - toolbar: si tendrá, o no, barra de herramientas.
  - location: si tendrá, o no, barra de dirección.
  - directories: si tendrá o no, botones Adelante/Atrás.
  - menubar: si tendrá, o no, barra de menú.
  - scrollbar: si tendrá, o no, barras de desplazamiento.
  - resizable: si se puede, o no, cambiar su tamaño.
  - width=px/height=px: ancho/alto en pixels.
  - left=px/top=px: posición izquierda/superior de la ventana.
  - opener: referencia a la ventana desde la que se abrió esta ventana.
- **.close()**: la cierra (pide confirmación, salvo que se haya abierto con open)
- **.moveTo(x,y)**: la mueve a las coordenadas indicadas. Esas coordenadas serán respecto de la posición de la esquina superior izquierda.
- **.moveBy(x,y)**: la desplaza los pixels indicados.
- **.resizeTo(x,y):** la da el ancho y alto indicados en pixels a una ventana creada con window.open().
- **.resizeBy(x,y)**: le añade ese ancho/alto.
- **.scroooX / scrollY**: scroll actual, horizontal / vertical, en pixels, de la ventana.

EJERCICIO 9

- Visualiza el ejemplo sobre el objeto window que se encuentra en el proyecto "Ejemplos BOM" en el Aula Virtual.
- Desde la consola:
  - Abre una nueva ventana de 500px X 300px.
  - Dicha ventana se abrirá en la posición 10,20.
  - Escribe en ella (con document.write()) un encabezado h1 que diga "Esto es una ventana emergente".
  - Muévela después 300px a la izquierda y 100 hacia abajo.
  - Tras esto último ciérrala.
- Continuando con la página creada en el ejercicio 8:
  - Incluye un nuevo botón que al ser pulsado abrirá una ventana emergente.
  - Dicha ventana contendrá, a su vez, un botón que al ser pulsado la cerrará.
  - La ventana tendrá el tamaño justo para incluir el botón y no tendrá barras de scroll, ni de herramientas, ni de dirección.

2.3. Propiedades localStorage y sessionStorage

Haciendo uso de diferentes mecanismos, es posible almacenar información del lado del cliente. Una de las opciones sencillas es el uso de WEB Storage API, que permite almacenar información de manera local en el navegador del usuario, sin necesidad de que sea enviada al servidor.

Las propiedades, del objeto Window, localStorage y sessionStorage permiten acceder a un objeto Storage, para almacenar datos en local. La diferencia entre estas dos propiedades es que localStorage almacena la información de forma indefinida (mientras no se limpien los datos del navegador, o se borren estos mediante JS) y en sessionStorage la información estará disponible mientras no se cierre la pestaña donde se esté utilizando.

La cantidad de información que se puede almacenar varía según el navegador (5Mb, 10Mb, etc.)

Como ya se ha comentado, BOM no está estandarizado por lo que es posible que el navegador no soporte esta API y, en consecuencia, no está de más asegurarse de ello:

```
if (typeof(Storage) !== "undefined") {
    // Código a ejecutar en caso de que el navegador soporte Storage
} else {
    // Código a ejecutar en caso contrario
}
```

Los datos a almacenar se registrarán en formato clave-valor, siendo ambos siempre de tipo String.

Tanto localStorage como sessionStorage, ofrecen, entre otros, las propiedades y métodos:

- length.
- getItem(clave): Devuelve el valor almacenado para la clave dada.
- setItem(clave, valor): Almacena el valor indicado asociado a la clave dada.
- removeItem(clave): Elimina el par clave-valor indicado.
- clear(): Limpia el objeto storage que corresponda.

Evento storage

Cuando se produce algún cambio en un objeto Storage, para localStorage, es lanzado el evento storage. Esto permite que, si distintas páginas del sitio web están compartiendo datos a través de localStorage, se sincronicen. Páginas que pertenezcan a distintos dominios no podrán acceder a los mismos objetos Storage.

2.4. Propiedad location

Propiedad de solo lectura, devuelve un objeto de tipo Location que contiene información sobre la URL actual del navegador y es posible modificarla. Sus principales propiedades y métodos son:

- **.href**: devuelve la URL actual completa
- **.protocol, .host, .port**: devuelve el protocolo, host y puerto respectivamente de la URL actual.
- **.reload()**: recarga la página actual. Se desencadena el mismo comportamiento que cuando se pulsa F5 o el botón del navegador que permite cargar de nuevo la página.
- **.assign(url)**: carga en el navegador la página pasada como parámetro.

2.5. Propiedad history

Referencia a un objeto de tipo History que permite acceder al historial de páginas visitadas y navegar por él:

- **.length**: muestra el número de páginas almacenadas en el historial.
- **.back()**: vuelve a la página anterior.
- **.forward()**: va a la siguiente página.
- **.go(x)**: siendo x un número, se mueve x páginas hacia adelante o hacia atrás (si x es negativo) en el historial.

2.6. Otras propiedades

Otros objetos que incluye BOM son:

- **document**: objeto ya conocido y trabajado en el apartado de DOM.
- **navigator**: da información sobre el navegador y el sistema en que se ejecuta. Entre otras opciones:
  - .userAgent: muestra información sobre el navegador que se está usando.
  - .plataform: muestra información sobre la plataforma sobre la que se ejecuta.
  - etc.
- **screen**: da información sobre la pantalla:
  - .width/.height: ancho/alto total de la pantalla (resolución)
  - .availWidth/.availHeight: igual pero excluyendo la barra del S.O.
  - etc.

EJERCICIO 10

- En la consola del navegador:
  - muestra la ruta completa de la página actual.
  - muestra el host de dicha página.
  - carga la página de Google a través del objeto location.
  - vuelve a la página anterior.
  - obtén todas las propiedades width/height y availWidth/availHeight del objeto screen. Compara el valor devuelto con el que devuelven las propiedades innerWidth/innerHeight y outerWidth/outerHeight de window.