Tabla de contenido

Interacción con los navegadores

Los siguientes objetos JS nos permiten interactuar con el navegador web:

- Screen
- Navigator
- Windows

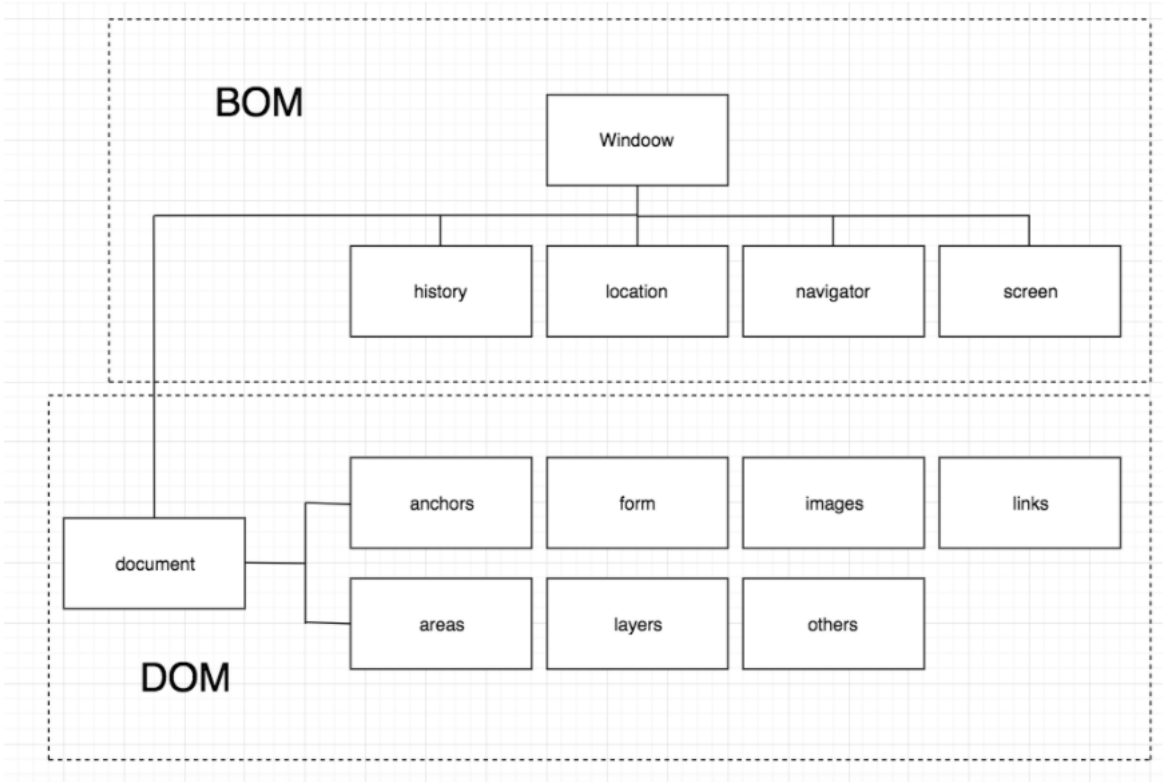
El principal problema de los objetos es que no hay ninguna norma que se encargue de su estandarización, con lo que históricamente los navegadores han implementado un modelo diferente y esto dificulta el trabajo de los programadores. Un ejemplo: en la mayoría de los navegadores los objetos **screen** y **navigator** también son propiedades de **window**. Como **window** es el objeto por defecto (superior en la jerarquía) no es necesario escribir su nombre. Así las líneas de código siguientes son equivalentes:

alert("hola") <-> window.alert("hola")
document.getElementById("nombre") <-> window.document.getElementById("edad")

La web http://www.w3schools.com/jsref/default.asp contiene una excelente guía de referencia de JavaScript con detalle de todos los objetos y métodos. Es interesante visitar los métodos que queremos emplear porque nos informan de en qué versiones de los navegadores están soportados.

Nosotros intentaremos trabajar con aquellos métodos que sean cross-browser, esto es, estén soportados en todos los navegadores.

Cuando se carga una página en un navegador se crean un número de objetos característicos del navegador según el contenido de la página.



La diferencia entre BOM y DOM

El modelo de objeto de navegador, BOM (Browser Object Model), se suele confundir con DOM pero no es lo mismo. Mientras el DOM se centraliza en el documento, el modelo BOM incluye **acceso a todas las áreas del navegador**. Otra gran diferencia es que el modelo **BOM es específico de cada navegador mientras que DOM pretende ser totalmente independiente del navegador** y se corresponde con el estándar definido por la W3C. Por el contrario para BOM, no hay entidad que se encargue de estandarizarlo o definir unos mínimos de interoperabilidad entre navegadores.

Para comprender el modelo DOM tenemos que analizar un documento HTML como si fuese un objeto formado por entes jerárquicos. Para el DOM todo el documento es un "nodo" de tipo documento y cada uno de los elementos dependientes son a su vez nodos hijos de este.

El enfoque del modelo DOM difiere totalmente del modelo BOM; en el modelo BOM nos movemos en base al nombre de los objetos (window, document, frames, etc.) en cambio el esquema de navegación del modelo DOM se basa en el enfoque jerárquico. Por eso, la navegación en un documento DOM es similar a recorrer una estructura en forma de árbol en donde cada objeto es un nodo y donde existe una jerarquía entre cada nodo. La ventaja de este enfoque es que podemos recorrer el documento sin saber nada previamente de su estructura.

Según esta estructura sabe siempre existe un nodo de nivel superior denominado nodo raíz y debajo de este nodo se desarrollan los nodos dependientes. Para una estructura de web común DOM necesita definir tres objetos base:

- **Node**: cada objeto del documento se define en un objeto Node. Existen distinto tipos de nodos.
- **NodeList**: es la lista de objetos Node.
- **NamedNodeMap**: este objeto nos permite acceder a todos los objetos Node utilizando el nombre en lugar de un índice

NOTA: se impartirá una unidad de trabajo sobre DOM, donde se desarrollará mejor estos objetos bases de DOM.

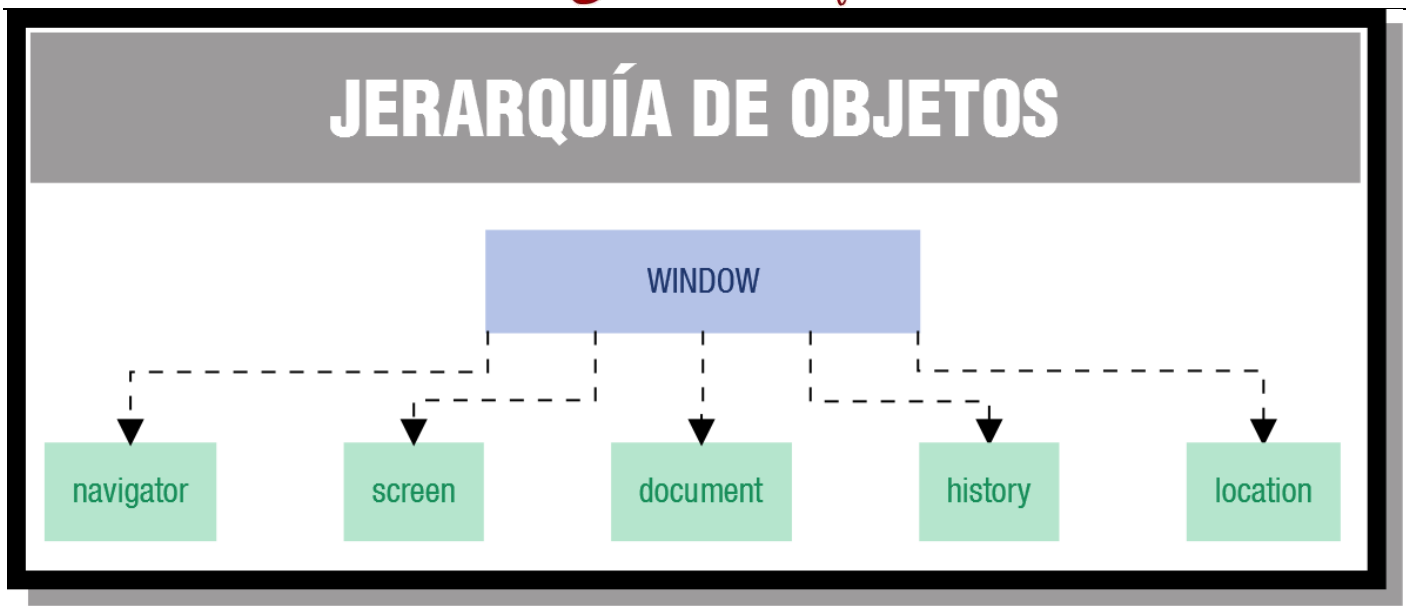
¿Qué es el DOM?

El Modelo de Objetos del Documento (DOM), permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos, sobre los que un programa de Javascript puede interactuar.

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

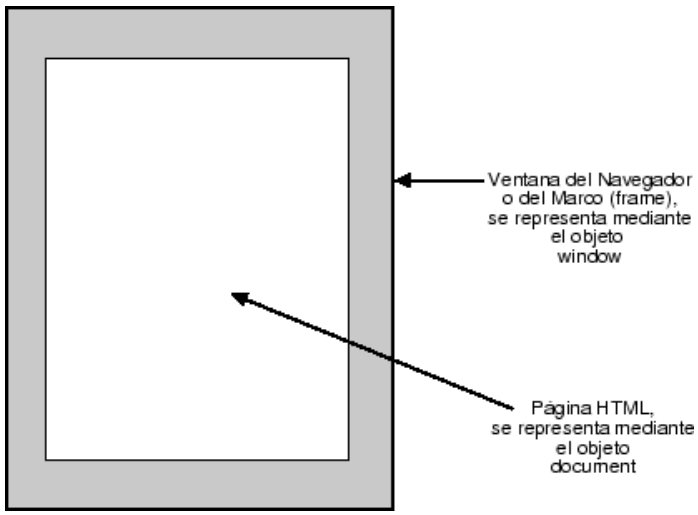
Según el W3C, el Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API), para documentos válidos HTML y bien construidos XML. Define la estructura lógica de los documentos, y el modo en el que se acceden y se manipulan.

objetos de alto nivel



El objeto *window* es el de más alto nivel, contiene las propiedades de la ventana y en el supuesto de trabajar con marcos (*frames*), se genera un objeto *window* para cada uno. El objeto *document* contiene todas las propiedades del documento actual, como son: su color de fondo, enlaces, imágenes, etc.

El objeto *navigator* contiene las propiedades del navegador. El objeto *location* contiene las propiedades de la URL activa. El objeto *history* contiene las propiedades que representan a las URL que el usuario ha visitado anteriormente. Es como una caché. El objeto *screen* contiene información referente a la resolución de la pantalla que muestra la URL.



Objeto navigator

El objeto Navigator contiene información sobre el propio navegador. Algunas propiedades del objeto Navigator más o menos estandarizadas son:

- **appCodeName**: Es el nombre en código del navegador.
- **appName**: Es el nombre oficial del navegador.
- **appVersion**: Es la versión del navegador.
- **plugins**: Es una matriz con los complementos del navegador.
- **userAgent**: Es el encabezado de agente de usuario del navegador.
- **cookieEnabled**: La propiedad cookieEnabled devuelve un valor booleano que especifica si las cookies están habilitadas en el navegador.
- **language**: devuelve la versión de idioma del navegador.

- **languages**: Devuelve una matriz de cadenas que representan los idiomas que conoce el usuario, por orden de preferencia.
- **onLine**: devuelve un valor booleano que especifica si el navegador está conectado a internet, flase sino lo está.
- **product**: Una cadena, que representa el nombre del motor del navegador
- **credentials** Devuelve la interfaz CredentialsContainer que expone métodos para solicitar credenciales y notificar al agente de usuario cuando ocurren eventos interesantes, como iniciar o cerrar sesión correctamente.
- **geolocation**: Devuelve un objeto Geolocalización que permite acceder a la ubicación del dispositivo (el navegador tiene que tener habilitada la geolocalización, normalmente en dispositivos móviles).
- **maxTouchPoints**: Devuelve el número máximo de puntos de contacto táctiles simultáneos admitidos por el dispositivo actual (pantallas táctiles).

Consulta más métodos y propiedades en la siguiente dirección, observa que hay algunos elementos que son experimentales y otros están en desuso (deprecated).

https://developer.mozilla.org/es/docs/Web/API/Navigator

Almacenamiento local y de sesión

Los navegadores utilizan el **almacenamiento local y el almacenamiento de sesión** para almacenar datos de forma permanente o temporal en el dispositivo del usuario.

El **almacenamiento local se utiliza para almacenar datos que deben persistir incluso cuando el navegador se cierra**. Por ejemplo, el almacenamiento local se puede utilizar para almacenar las preferencias de visualización de un usuario, los datos de inicio de sesión o el contenido de un carrito de compras.

El almacenamiento de sesión se utiliza para almacenar datos que solo deben persistir durante la sesión actual del navegador. Por ejemplo, el almacenamiento de sesión se puede utilizar para almacenar el idioma preferido del usuario, el estado de un carrito de compras o el estado de inicio de sesión.

En concreto, los navegadores utilizan el almacenamiento local y el almacenamiento de sesión para los siguientes propósitos:

- **Almacenar preferencias de usuario**: Los navegadores utilizan el almacenamiento local para almacenar las preferencias de usuario, como el idioma preferido, el tamaño de fuente y la configuración de privacidad. Estas preferencias se utilizan para personalizar la experiencia del usuario en el navegador.
- **Almacenar datos de inicio de sesión**: Los navegadores utilizan el almacenamiento local para almacenar los datos de inicio de sesión del usuario, como el nombre de usuario y la contraseña. Esto permite a los usuarios iniciar sesión en sitios web y aplicaciones sin tener que escribir sus credenciales cada vez.
- **Almacenar datos de seguimiento**: Los navegadores utilizan el almacenamiento local para almacenar datos de seguimiento, como los sitios web que ha visitado el usuario. Estos datos se pueden utilizar para personalizar la publicidad o para mejorar la experiencia del usuario.
- **Almacenar datos temporales**: Los navegadores utilizan el almacenamiento de sesión para almacenar datos temporales, como el contenido de un carrito de compras o el estado de un juego en línea. Estos datos se eliminan cuando el usuario cierra la pestaña o la ventana del navegador.

Es importante tener en cuenta que los navegadores pueden eliminar datos del almacenamiento local o del almacenamiento de sesión si el dispositivo del usuario está bajo presión de almacenamiento. Por lo tanto, es importante utilizar el almacenamiento de forma eficiente para evitar perder datos importantes.

La cantidad de espacio disponible para utilizar como almacenamiento con la propiedad **navigator.storage** depende del navegador y del sistema operativo. Actualmente, los siguientes son los límites de almacenamiento para los navegadores más populares:

- **Chrome**: 5MB para el almacenamiento local y 10MB para el almacenamiento de sesión
- **Firefox**: 5MB para el almacenamiento local y 10MB para el almacenamiento de sesión
- **Safari**: 5MB para el almacenamiento local y 10MB para el almacenamiento de sesión
- **Edge**: 5MB para el almacenamiento local y 10MB para el almacenamiento de sesión

Es importante tener en cuenta que estos límites pueden variar según la versión del navegador y el sistema operativo. Además, los navegadores pueden eliminar datos del almacenamiento si el dispositivo del usuario está bajo presión de almacenamiento.

El objeto **navigator.storage** utiliza ambos el almacenamiento local y el almacenamiento de sesión. La propiedad navigator.storage es una instancia de la clase StorageManager. Esta clase proporciona una serie de métodos y propiedades para acceder y manipular el almacenamiento.

Para acceder al almacenamiento local, puedes utilizar el método open() con el nombre localStorage.

```
// Obtiene el objeto StorageManager
const storageManager = navigator.storage;

// Obtiene el almacenamiento local
const storage = storageManager.open("localStorage");
```

Para acceder al almacenamiento de sesión, puedes utilizar el método open() con el nombre sessionStorage.

```
// Obtiene el objeto StorageManager
const storageManager = navigator.storage;

// Obtiene el almacenamiento de sesión
const storage = storageManager.open("sessionStorage");
```

En resumen, el objeto navigator.storage es una capa de abstracción que permite acceder a ambos tipos de almacenamiento. Aquí hay un ejemplo de cómo utilizar el objeto navigator.storage para almacenar el nombre del usuario en el almacenamiento local y el idioma preferido del usuario en el almacenamiento de sesión:

```
// Obtiene el objeto StorageManager
const storageManager = navigator.storage;

// Obtiene el almacenamiento local
const storage = storageManager.open("localStorage");

// Almacena el nombre del usuario en el almacenamiento local
storage.setItem("name", "Juan");
```

```
// Obtiene el almacenamiento de sesión
const sessionStorage = storageManager.open("sessionStorage");

// Almacena el idioma preferido del usuario en el almacenamiento de sesión
sessionStorage.setItem("language", "es");
```

Este código escribirá el nombre "Juan" en el almacenamiento local y el idioma "es" en el almacenamiento de sesión.
Para acceder y borrar los elementos almacenados utilizamos

```
// Lee un dato del almacenamiento
const nombre = storage.getItem("name");

// Borra un dato del almacenamiento
storage.removeItem("name");
```

Actividad 1

1. Define un código javascript que te permita obtener la longitud y latitud del dispositivo desde donde se muestra la página. Trate de ejecutar el mismo código en distintos navegadores para detectar diferencias en el comportamiento.

2. Define un código javascript que muestre cuántos Gigas de RAM dispone el dispositivo y cuántos núcleos/procesadores tiene. Trate de ejecutar el mismo código en distintos navegadores para detectar diferencias en el comportamiento.

3. Define un código javascript que detecte si se dispone de una webcam y de un dispositivo de salida de audio. Se mostrará un mensaje tanto si se tiene como si no. Trate de ejecutar el mismo código en distintos navegadores para detectar diferencias en el comportamiento.

4. Lee la siguiente información sobre los comentarios condicionales para Internet Explorer y conteste a las siguientes cuestiones
   http://librosweb.es/libro/css_avanzado/capitulo_6/comentarios_condicionales_filtros_y_hacks.html
   a. Indique cómo sería el código para cargar la hoja de estilos ESTILO1.css para el internet explorer 5 o 7
   b. Y la hoja ESTILO2.css para internet explorer 9.

Objeto Screen

El objeto screen se utiliza para obtener información sobre la pantalla del usuario. Uno de los datos más importantes que proporciona el objeto screen es la resolución del monitor en el que se están visualizando las páginas. Los diseñadores de páginas web necesitan conocer las resoluciones más utilizadas por los usuarios para adaptar sus diseños a estas resoluciones.

Las siguientes propiedades están disponibles en el objeto screen:

- **availHeight**: Altura de pantalla disponible para las ventanas
- **availWidth**: Anchura de pantalla disponible para las ventanas
- **colorDepth**: Profundidad de color de la pantalla (32 bits normalmente)
- **height**: Altura total de la pantalla en píxel

- **width**: Anchura total de la pantalla en píxel

La altura/anchura de pantalla disponible para las ventanas es menor que la altura/anchura total de la pantalla, ya que se tiene en cuenta el tamaño de los elementos del sistema operativo como por ejemplo la barra de tareas y los bordes de las ventanas del navegador.

Además de la elaboración de estadísticas de los equipos de los usuarios, las propiedades del objeto screen se utilizan por ejemplo para determinar cómo y cuánto se puede redimensionar una ventana y para colocar una ventana centrada en la pantalla del usuario.

Actividad 2

Defina un código javascript donde muestre todas las propiedades que se han indicado sobre el objeto Screen. ¿Cuál es mayor height o availHeight? ¿Cuál es la más recomendable utilizar?

Objeto Windows

Está asociado a la ventana del navegador. Dado que es el objeto por defecto no es necesario nombrarlo. Muchas de las utilidades que ya conoces como la función alert() o la función prompt() pertenecen a este objeto. También incluye otros objetos importantes como document (parte visible de la página web), history (historial del navegador) o location (url del navegador).

El objeto window permite gestionar ventanas. Así dispone de métodos para crear nuevas ventanas, modificar las existentes o cerrarlas. Los frames también son objetos que pertenecen a window. Veamos primero sus propiedades o atributos:

- **closed**: Propiedad booleana que indica si la ventana ha sido cerrada o no
- **document**: OBJETO document
- **frames**: ARRAY con todos los objetos frame (incluyendo iframes) de la ventana actual
- **history**: OBJETO history
- **innerHeight**: Permite leer/escribir el alto del área de contenido de la ventana excluyendo las barras de herramientas y las áreas de scrolling. No soportado en IE8 y anteriores.
- **innerWidth**: Permite leer/escribir el ancho del área de contenido de la ventana excluyendo las barras de herramientas y las áreas de scrolling. No soportado en IE8 y anteriores.
- **length**: Devuelve el número de frames (incluyendo iframes) en una ventana
- **location**: OBJETO location
- **name**: Permite leer/escribir un nombre asociado a esta ventana
- **navigator**: OBJETO navigator (a menudo se considera que pertenece a window)
- **opener**: Retorna el nombre de la ventana que abrió o creó la ventana actual.
- **outerHeight**: Permite leer/escribir el alto del área de contenido de la ventana que sí incluye las barras de herramientas y las áreas de scrolling. No soportado en IE8 y anteriores. outerWidth Permite leer/escribir el ancho del área de contenido de la ventana que sí incluye las barras de herramientas y las áreas de scrolling. No soportado en IE8 y anteriores.
- **pageXOffset**: Almacena el número de pixels en la horizontal que se ha desplazado mediante scrolls el document actual (desde la esquina superior izquierda de la ventana). En versiones anteriores a IE8 usar document.body.scrollLeft
- **pageYOffset**: Almacena el número de pixels en la vertical que se ha desplazado mediante scrolls el document actual (desde la esquina superior izquierda de la ventana). En versiones anteriores a IE8 usar document.body.scrollTop
- **Parent**: Devuelve una referencia a la ventana padre de la actual

- **Screen:** OBJETO screen (a menudo se considera que pertenece a window)
- **screenLeft**: Devuelve la coordenada x (horizontal) de esta ventana en relación a la pantalla o monitor Ojo: en Firefox no se reconoce, se debe usar screenX
- **screenTop:** Devuelve la coordenada y (vertical) de esta ventana en relación a la pantalla o monitor Ojo: en Firefox no se reconoce, se debe usar screenY
- **screenX**: Devuelve la coordenada x (horizontal) de esta ventana en relación a la pantalla o monitor Ojo: en IE no se reconoce, se debe usar screenLeft
- **screenY**: Devuelve la coordenada y (vertical) de esta ventana en relación a la pantalla o monitor Ojo: en IE no se reconoce, se debe usar screenTop
- **Self**: Devuelve una referencia a la ventana actual
- **Status**: Permite grabar un texto en la barra de estado de la ventana Puede requerir (salvo en Ópera) que haya que configurarlo en los ajustes del navegador
- **Top**: Devuelve una referencia a la ventana superior en la jerarquía

Y ahora presentemos los métodos del objeto window

- **alert(mensaje):** Muestra un mensaje en una ventana de alerta con botón Aceptar
- **blur():** Quita el foco (atención del cursor) de la ventana actual No soportado en algunas versiones de Opera
- **clearInterval(vble)** Borra el lanzamiento automático colocado con setInterval(func, millsg)
- **clearTimeout()** Borra el lanzamiento automático colocado con setTimeout(func, millsg)
- **close():** Cierra la ventana actual
- **confirm():** Método booleano que muestra un mensaje con botones Aceptar/Cancelar
- **createPopup():** Crea una ventana emergente. Atención: sólo funciona en IE, no usar.
- **focus():** Lo contrario a blur(), coloca el foco (atención del cursor) en la ventana. En este caso sí que está soportado en Ópera.
- **moveBy(x, y):** Mueve una ventana desde su posición actual de forma relativa. Es decir, desplaza x en horizontal e y en vertical. Se recomienda hacer focus(), si no la ventana puede perder el foco y no ser visible. *Si la ventana contiene una url que no pertenece al mismo directorio-dominio que la ventana madre podemos obtener un error de tipo Error: Permission denied to access property 'moveBy'*
- **moveTo(x,y):** Mueve una ventana a una posición absoluta. Es decir la coloca en las coordenadas x (para horizontal) e y (para vertical)
- **open():** Métodos para abrir nuevas ventanas. Lo veremos en detalle.
- **print()** Imprime el contenido de la ventana actual
- **prompt(mensaje, valor por defecto):** Muestra un cuadro de diálogo para recoger una entrada sencilla del usuario. resizeBy(x,y) Cambia el tamaño de la ventana de forma relativa. X e y pueden ser números positivos o negativos y al ancho y alto actuales se les sumará o restará el valor de x e y. No soportado en algunas versiones de Opera y Chrome.
- **resizeTo(x,y):** Cambia el tamaño de forma absoluta. El nuevo tamaño será x píxeles de ancho por y píxeles de alto. Soportado en todos los navegadores.
- ~~**scroll()**~~ OBSOLETO. Usar **scrollBy**() o **scrollTo**()
- **scrollBy(x,y):** Desplaza el contenido una cantidad horizontal de x píxeles y una cantidad vertical de y píxeles desde la posición actual.
- **scrollTo(x,y):** Desplaza el contenido a la posición x (horizontal) e y (vertical)
- **setInterval(func, msg):** Ejecuta la función func en intervalos de millsec milisegundos
- **setTimeout(func, millsec):** Ejecuta una function o evalúa una expression una vez después de que hayan transcurrido millsec milisegundos.

Trabajo con ventanas.

JS permite, mediante el método open del objeto window, trabajar con diferentes ventanas del navegador

window.open(URL, nombre, características, reemplazar)

URL: Es un parámetro opcional: especifica la URL que queremos cargar en la nueva ventana. Si no se indica se abrirá una ventana vacía (blank).

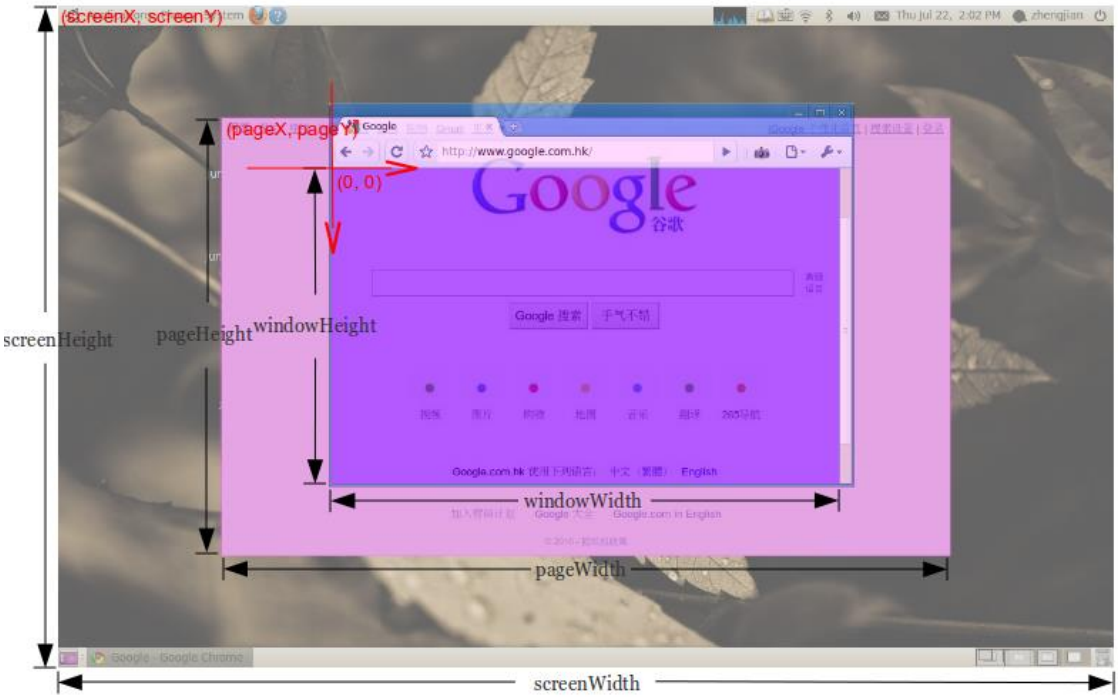
Nombre: También es un parámetro opcional. Se utiliza si desde JS necesitamos tener referenciada la ventana para, por ejemplo, abrir enlaces en ella (mediante el atributo target de la etiqueta <a>). Como bien sabes el atributo target de <a> puede adquirir los siguientes valores:

- **_blank** valor por defecto. El enlace se abre en una ventana nueva.
- **_parent** El enlace se carga en el marco padre (trabajo con marcos)
- **_self** El enlace se carga en la ventana/marco actual.
- **_top** El enlace se carga en la ventana origen de todos los marcos actuales. Los marcos se eliminarán.
- **nombre** El enlace se carga en la ventana con este nombre.

Características: Se trata de una cadena opcional de valores separados por comas. Permite configurar la ventana con atributos. Valores aceptados (todos opcionales) son los siguientes:

- **channelmode=yes|no|1|0** Indica cuándo mostrar la ventana en modo teatro. Por defecto su valor es no. Sólo para IE.
- **directories=yes|no|1|0** Indica cuándo mostrar botones para carpetas y directorios. Por defecto es yes. Sólo para IE.
- **fullscreen=yes|no|1|0** Indica cuándo mostrar la ventana en pantalla completa. Por defecto es no. Sólo para IE.
- **height=pixels** Alto de la ventana. El valor mínimo aceptado es 100.
- **left=pixels** Posición izquierda del origen de coordenadas.
- **location=yes|no|1|0** Cuándo mostrar la barra de URL. Por defecto es yes.
- **menubar=yes|no|1|0** Cuándo mostrar la barra de menús. Por defecto es yes.
- **resizable=yes|no|1|0** Permitir que el usuario cambie el tamaño. Por defecto yes
- **scrollbars=yes|no|1|0** Cuando mostrar las barras de scroll. Por defecto yes
- **status=yes|no|1|0** Cuando agregar barra de estado. Por defecto yes
- **titlebar=yes|no|1|0** Cuando mostrar la barra de título. Generalmente se ignora cuando la aplicación que llama a esta ventana no es una página web o un cuadro de diálogo. Por defecto yes.
- **toolbar=yes|no|1|0** Cuando mostrar la barra de herramientas del navegador. Por defecto yes
- **top=pixels** Posición superior del origen de coordenadas. Sólo IE
- **width=pixels** Ancho de la ventana. Valor mínimo de 100 píxeles.

Reemplazar: Valor booleano opcional. Si es true la URL reemplaza al documento actual en la lista del historial del navegador. Si por el contrario es false la URL se graba en el historial como una nueva entrada.



Ventana principal y ventana secundaria

Al trabajar con ventanas tenemos que diferenciar entre ventana principal (aquella que llama al método open) y la ventana secundaria (aquella que se ha creado).

Desde la ventana principal podemos obtener una referencia a la secundaria:

```
let secundaria = window.open("","", "location=no, menubar=no");
if (secundaria == null)
      alert("La creación de la ventana secundaria ha fallado");
else
      secundaria.document.write("<html><head></head><body>Hola</body></html>");
```

En otras ocasiones desearemos abrir una ventana secundaria automáticamente. Veamos como quedaría este código:

```
function ventanaSecundaria (URL){
   window.open(URL,"ventana1","width=120,height=300,scrollbars=NO")
}
```

Objeto history.

Este objeto guarda una colección de URLS visitadas por una ventana concreta del navegador, es decir, su historial. Se trata de un objeto que es a su vez una propiedad del objeto window.

PROPIEDADES

length: Guarda la longitud del historial.
current: URL actual (propiedad privada no accesible desde el exterior)
next: Siguiente entrada en el historial (propiedad no accesible)
previous: Entrada anterior en el historial (propiedad no accesible)

MÉTODOS

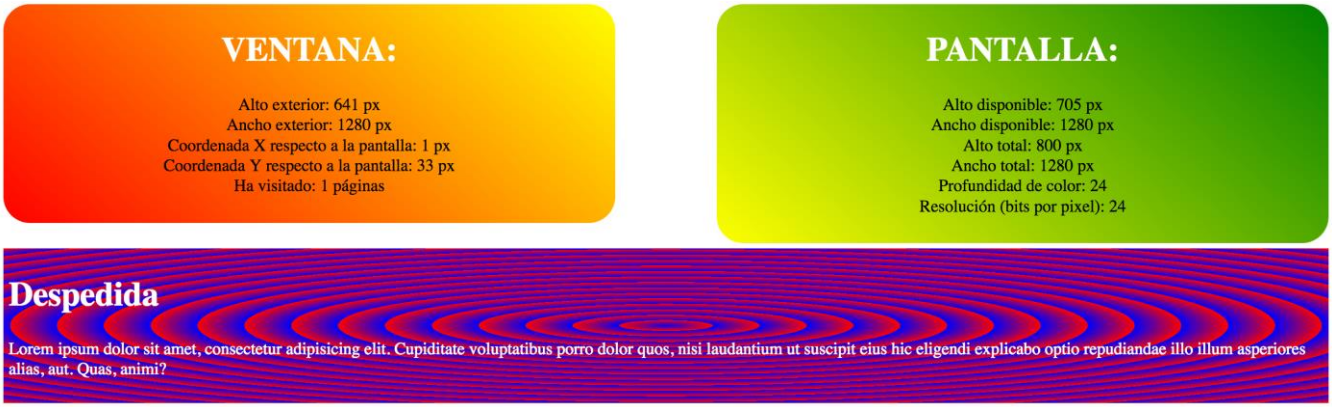
back(): Carga en la ventana actual la entrada anterior en el historial.
forward(): Carga en la ventana actual la entrada siguiente en el historial.
go(x): Nos permite movernos por el historial. Si x es un número positivo nos moveremos hacia adelante. Si x es negativo hacia atrás. history.go(-1) equivale a history.back().

Recuerda que siempre es preferible utilizar los métodos de un objeto antes que utilizar directamente sus propiedades. Algunos navegadores no permitirán el acceso directo a las propiedades. De hecho, muchos tutoriales de JS en la red no mostrarán siquiera las propiedades current, next y previous.

Actividad 3

Cree la siguiente página, han de utilizarse al menos 3 div, uno para cada bloque: pantalla, ventana y despedida. La información que se muestra en el bloque pantalla se ha obtenido con JavaScript al cargar la página con objeto screen y el bloque ventana con objeto windows.

Cree otro fichero CSS, cuando el navegador que abra esta misma página sea Mozilla Firefox, se cargará este último CSS y no el anterior y se verá tal que:



Objeto location.

Este objeto, también propiedad de window, nos permite acceder a la URL cargada en la ventana actual. Uno de los usos más habituales consiste en cargar mediante JS nuevas páginas en una ventana. También nos va a permitir acceder a cada una de las partes de la URL cargada como puede ser el protocolo, el puerto, el dominio, etc.

PROPIEDADES

Veamos sobre un ejemplo de URL: http://www.midominio.com:8080/libro.htm#capitulo3

hash: Si en la URL hay una parte de enlace la devuelve. En el ejemplo devuelve #capitulo3
host: Retorna el nombre de máquina y el puerto, en caso de que existan en la URL. En el ejemplo anterior retorna: www.midominio.com:8080
hostname: Retorna sólo la parte del nombre de máquina. En el ejemplo www.midominio.com
href: Retorna la URL entera
pathname: Retorna la ruta de la URL. Es igual a la URL entera pero quitando la parte de hash.
port: Sólo si se indica en la URL retorna el puerto.
protocol: Retorna sólo el protocolo utilizado. En el ejemplo retorna http:
search: Retorna la porción de consulta de la url. Esto es, los parámetros de tipo get que tienen la forma ?param1=valor1&param2=valor2&…

MÉTODOS

assign(**URL**): Carga una nueva URL en la ventana
reload(): Recarga la URL actual en la ventana (equivale a actualizar – F5)
replace(**URL**): Reemplaza la URL actual por otra.

Actividad 4

Cree una página web donde aparezcan los siguientes botones:

Abrir Ventana: abrirá una ventana de tamaño 400px*400px con la dirección www.google.es. La nueva ventana deberá aparecer centrada en la pantalla.

Cerrar Ventana: deberá cerrar la ventana que se acaba de crear. Si la ventana aún no se ha creado o bien el usuario la cerró, deberá mostrar un error que diga "No hay ventana que cerrar".

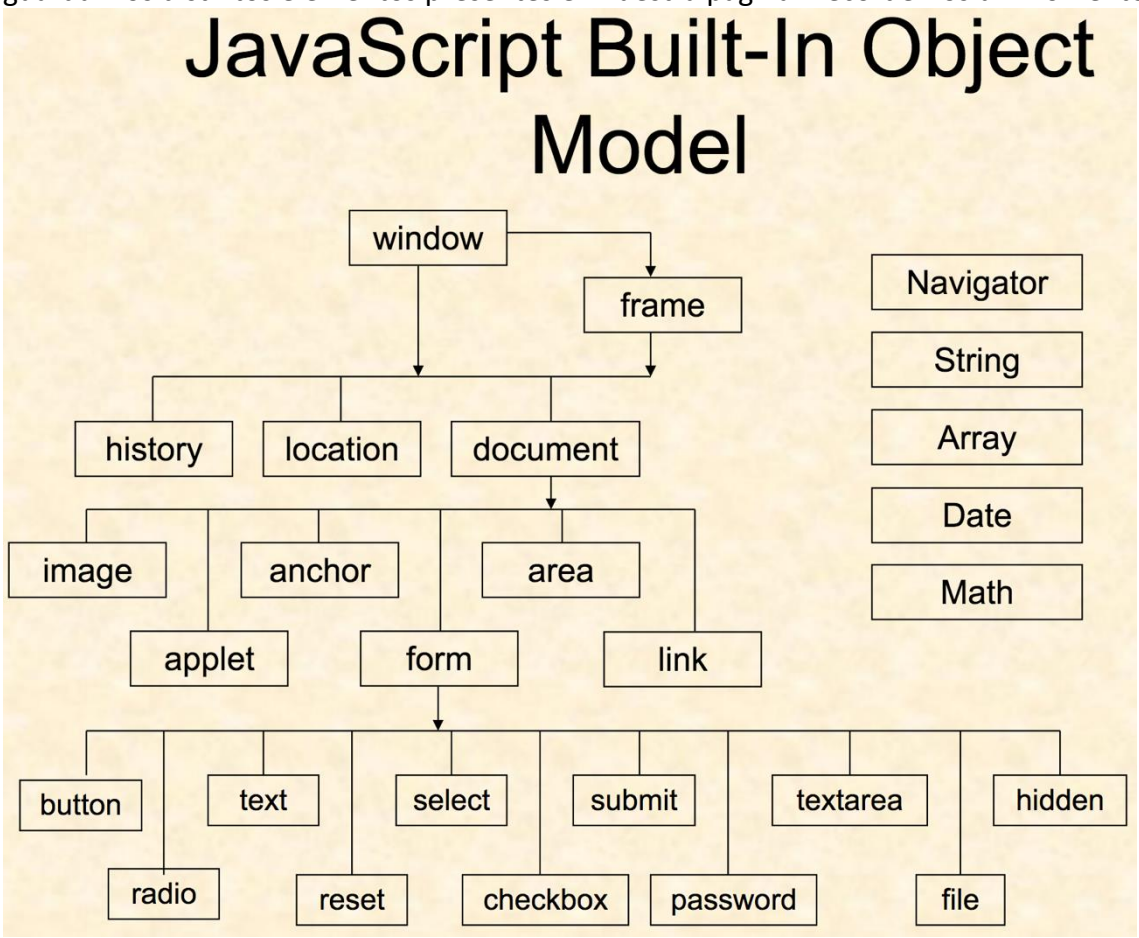
Ampliar: modificará el tamaño de la ventana aumentando tanto el ancho como en 30px. Se mostrará un mensaje de error cuando no se pueda ampliar el alto o el ancho. NOTA: para que funcione este y los siguientes apartados tiene que abrir una ventana en blanco, para ello quite la dirección www.google.es

Reducir: modificará el tamaño de la ventana disminuyendo tanto el ancho como en 30px. Se mostrará un mensaje de error cuando no se pueda reducir el alto o el ancho.

Mover: Ubicará la pantalla en la posición indicada en los cuadros de texto. Los valores tendrán que ser enteros positivos. Antes de mover la nueva pantalla tendrá que comprobar que la nueva posición de la ventana es posible teniendo en cuenta tanto la posición como su alto y ancho y el tamaño de la pantalla.

Objeto document y descendientes.

Este objeto representa el documento cargado en una ventana, es decir, la parte visible de contenido de la misma. Es el objeto que nos permite acceder a cada uno de los controles y elementos de nuestra página web. Se trata de una propiedad del objeto window. El objeto document, además, contiene colecciones que guardan los distintos elementos presentes en nuestra página. Recordemos un momento la jerarquía básica:



Recuerda que en el caso de que existan varios formularios o imágenes o applets o enlaces en una página éstos son accesibles como colecciones. Por ejemplo si existen varios formularios puedo acceder al primero de la siguiente forma:

let formulario = document.form[0];

Y si este formulario tiene nombre (atributo name) podría acceder también mediante esta sintaxis (es la que hemos utilizado a menudo):

var formulario = document.nombre;

Para una referencia completa del objeto document puedes visitar esta página de w3schools: http://www.w3schools.com/jsref/dom_obj_document.asp

JSON

El objeto JSON de JavaScript es un objeto global que proporciona una serie de métodos y propiedades para trabajar con datos JSON. JSON significa JavaScript Object Notation. Es un formato de datos basado en texto que se utiliza para representar datos estructurados.

Los datos JSON se representan como pares clave-valor. La clave es un nombre, que debe ser una cadena de texto, y el valor puede ser cualquier tipo de dato, incluyendo cadenas de texto, números, booleanos, arrays y objetos. Ejemplo de JSON:

```
{
  "name": "Juan",
  "age": 30,
  "is_married": true,
  "children": [
    {
      "name": "María",
      "age": 20
    },
    {
      "name": "Pedro",
      "age": 10
    }
  ]
}
```

En este ejemplo, la clave "name" tiene un valor de cadena de texto, la clave "age" tiene un valor numérico, la clave "is_married" tiene un valor booleano y la clave "children" tiene un valor de arreglo.

Reglas de sintaxis de JSON:

Las claves deben ser definidas entre comillas dobles (") y tienen que ser cadenas de texto.

Los valores pueden ser cualquier tipo de dato, incluyendo cadenas de texto, números, booleanos, arreglos y objetos.

Los valores deben estar separados por comas.

Los objetos deben estar encerrados entre llaves ({}).

Los arrays deben estar encerrados entre corchetes ([]).

Para trabajar con JSON en JavaScript, puede utilizar los métodos **JSON.parse()** y **JSON.stringify()**. El método JSON.parse() convierte una cadena JSON en un objeto JavaScript, mientras que el método JSON.stringify() convierte un objeto JavaScript en una cadena JSON. Aquí hay un ejemplo de cómo utilizar estos métodos:

```
// Convertir una cadena JSON en un objeto JavaScript
var persona = '{"nombre":"Juan","edad":30,"ciudad":"Madrid"}';
var personaObjeto = JSON.parse(persona);

// Convertir un objeto JavaScript en una cadena JSON
```

```
var persona = { nombre: "Juan", edad: 30, ciudad: "Madrid" };
var personaCadena = JSON.stringify(persona);
```

Actividad 5

Defina una página web para gestionar un carrito de la compra. El carrito será array de productos. Los productos serán JSON tienen que tener: nombre, cantidad y precio.

Utilice la memoria local para mantener guardado el carrito.

La aplicación deberá crear un carrito (error si ya tiene uno creado), mostrar el contenido del carrito, añadir/quitar productos y borrar el carrito.

Verifique que al cerrar la ventana puede recuperar el carrito almacenado en la memoria local.

Fuentes

http://www.maestrosdelweb.com/que-es-javascript/
http://librosweb.es/libro/javascript/
http://librosweb.es/
http://www.aprenderaprogramar.es/index.php?option=com_content&view=article&id=788:tipos-de-datos-javascript-tipos-primitivos-y-objeto-significado-de-undefined-null-nan-ejemplos-cu01112e-&catid=78:tutorial-basico-programador-web-javascript-desde-&Itemid=206
https://www.todojs.com/ref/javascript/global/
https://jherax.wordpress.com/2014/07/08/javascript-poo-1/
https://ikastaroak.ulhi.net/edu/es/DAW/DWEC/DWEC03/es_DAW_DWEC03_Contenidos/website_1_objetos_de_ms_alto_nivel_en_javascript.html