PRÁCTICA 11: MANEJO DE BOM (BROWSER OBJECT MODEL)

Duración estimada: 150 min.

Objetivos:

- Manejar propiedades del navegador con JavaScripts (BOM)

Bibliografía: ***Internet***

BROWSER OBJECT MODEL

1. Introducción a BOM:

Las versiones 3.0 de los navegadores Internet Explorer y Netscape Navigator introdujeron el concepto de Browser Object Model o BOM, que permite acceder y modificar las propiedades de las ventanas del propio navegador.

Mediante BOM, es posible redimensionar y mover la ventana del navegador, modificar el texto que se muestra en la barra de estado y realizar muchas otras manipulaciones no relacionadas con el contenido de la página HTML.

- Crear, mover, redimensionar y cerrar ventanas de navegador.
- Obtener información sobre el propio navegador.
- Propiedades de la página actual y de la pantalla del usuario.
- Gestión de cookies.
- Objetos ActiveX en Internet Explorer.

2. El objeto window:

El objeto **window** representa la ventana completa del navegador. Mediante este objeto, es posible mover, redimensionar y manipular la ventana actual del navegador. Incluso es posible abrir y cerrar nuevas ventanas de navegador.

Como todos los demás objetos heredan directa o indirectamente del objeto window, no es necesario indicarlo de forma explícita en el código JavaScript. En otras palabras:

window.forms[0] === forms[0]

window.document === document

BOM define cuatro métodos para manipular el tamaño y la posición de la ventana:

- **moveBy(x, y)** desplaza la posición de la ventana x píxel hacia la derecha y y píxel hacia abajo. Se permiten desplazamientos negativos para mover la ventana hacia la izquierda o hacia arriba.
- **moveTo(x, y)** desplaza la ventana del navegador hasta que la esquina superior izquierda se encuentre en la posición (x, y) de la pantalla del usuario. Se permiten desplazamientos negativos, aunque ello suponga que parte de la ventana no se visualiza en la pantalla.
- r**esizeBy(x, y)** redimensiona la ventana del navegador de forma que su nueva anchura sea igual a (anchura_anterior + x) y su nueva altura sea igual a (altura_anterior + y). Se pueden emplear valores negativos para reducir la anchura y/o altura de la ventana.
- **resizeTo(x, y)** redimensiona la ventana del navegador hasta que su anchura sea igual a x y su altura sea igual a y. No se permiten valores negativos.

Los navegadores son cada vez menos permisivos con la modificación mediante JavaScript de las propiedades de sus ventanas. De hecho, la mayoría de navegadores permite a los usuarios bloquear el uso de JavaScript para realizar cambios de este tipo. De esta forma, una aplicación nunca debe suponer que este tipo de funciones están disponibles y funcionan de forma correcta.

Además de desplazar y redimensionar la ventana del navegador, es posible averiguar la posición y tamaño actual de la ventana. Sin embargo, la ausencia de un estándar para BOM provoca que cada navegador implemente su propio método:

Internet Explorer proporciona las propiedades **window.screenLeft y window.screenTop** para obtener las coordenadas de la posición de la ventana. No es posible obtener el tamaño de la ventana completa, sino solamente del área visible de la página (es decir, sin barra de estado ni menús). Las propiedades que proporcionan estas dimensiones son d**ocument.body.offsetWidth** y **document.body.offsetHeight.**

Los navegadores de la familia **Mozilla**, **Safari** y **Opera** proporcionan las propiedades **window.screenX** y **window.screenY** para obtener la posición de la ventana. El tamaño de la zona visible de la ventana se obtiene mediante **window.innerWidth** y

window.innerHeight, mientras que el tamaño total de la ventana se obtiene mediante **window.outerWidth** y **window.outerHeight**.

3. Control de tiempos:

Al contrario que otros lenguajes de programación, JavaScript no incorpora un método **wait()** que detenga la ejecución del programa durante un tiempo determinado. Sin embargo, JavaScript proporciona los métodos **setTimeout()** y **setInterval()** que se pueden emplear para realizar tareas similares.

El método **setTimeout()** permite ejecutar una función al transcurrir un determinado periodo de tiempo:

```
setTimeout("console.log('Han transcurrido 3 segundos desde que me programaron')", 3000);

function muestraMensaje() {
  console.log("Han transcurrido 3 segundos desde que me programaron");
}

setTimeout(muestraMensaje, 3000);
```

Como es habitual, cuando s***e indica la referencia a la función no se incluyen los paréntesis***, ya que de otro modo, ***se ejecuta la función en el mismo instante en que se establece el intervalo de ejecución.***

Cuando se establece una cuenta atrás, la función **setTimeout()** devuelve el identificador de esa nueva cuenta atrás. Empleando ese identificador y la función **clearTimeout()** es posible impedir que se ejecute el código pendiente:

```
function muestraMensaje() {
  console.log("Han transcurrido 3 segundos desde que me programaron");
}
var id = setTimeout(muestraMensaje, 3000);


// Antes de que transcurran 3 segundos, se decide eliminar la ejecución pendiente

clearTimeout(id);
```

Además de programar la ejecución futura de una función, JavaScript también permite establecer la ejecución periódica y repetitiva de una función. El método necesario es **setInterval()** y su funcionamiento es idéntico al mostrado para **setTimeout()**:

```
function muestraMensaje() {
  console.log("Este mensaje se muestra cada segundo");
}
setInterval(muestraMensaje, 1000);
```

De forma análoga a ***clearTimeout()***, también existe un método que permite eliminar una repetición periódica y que en este caso se denomina ***clearInterval()***:

```
function muestraMensaje() {
  console.log("Este mensaje se muestra cada segundo");
}
var id = setInterval(muestraMensaje, 1000);

// Despues de ejecutarse un determinado número de veces, se elimina el intervalo

clearInterval(id);
```

4. El objeto document:

El objeto document es el único que pertenece tanto al DOM (como se vio en el capítulo anterior) como al BOM. Desde el punto de vista del BOM, el objeto document proporciona información sobre la propia página HTML.

Algunas de las propiedades más importantes definidas por el objeto document son:

Propiedad	Descripción
lastModified	La fecha de la última modificación de la página
referrer	La URL desde la que se accedió a la página (es decir, la página anterior en el array history)
title	El texto de la etiqueta <title>
URL	La URL de la página actual del navegador

Array	Descripción
anchors	Contiene todas las "anclas" de la página (los enlaces de tipo)
applets	Contiene todos los applets de la página
embeds	Contiene todos los objetos embebidos en la página mediante la etiqueta <embed>
forms	Contiene todos los formularios de la página
images	Contiene todas las imágenes de la página
links	Contiene todos los enlaces de la página (los elementos de tipo)

Los elementos de cada array del objeto document se pueden acceder mediante su índice numérico o mediante el nombre del elemento en la página HTML. Si se considera por ejemplo la siguiente página HTML:

```
<html>
  <head><title>Pagina de ejemplo</title></head>
  <body>
    <p>Primer parrafo de la pagina</p>
    <a href="otra_pagina.html">Un enlace</a>
    <img src="logo.jpg" name="logotipo"/>
    <form method="post" name="consultas">
      <input type="text" name="id" />
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Para acceder a los elementos de la página se pueden emplear las funciones DOM o los objetos de BOM:

Párrafo: document.getElementsByTagName("p")

Enlace: document.links[0]

Imagen: document.images[0] o document.images["logotipo"]

Formulario: document.forms[0] o document.forms["consultas"]

Una vez obtenida la referencia al elemento, se puede acceder al valor de sus atributos HTML utilizando las propiedades de DOM. De esta forma, el método del formulario se obtiene mediante document.forms["consultas"].method y la ruta de la imagen es document.images[0].src.

5. El objeto location:

El objeto **location** es uno de los objetos más útiles del BOM. Debido a la falta de estandarización, **location** es una propiedad tanto del objeto window como del objeto document. El objeto **location** representa la URL de la página HTML que se muestra en la ventana del navegador y proporciona varias propiedades útiles para el manejo de la URL:

Propiedad	Descripción
hash	El contenido de la URL que se encuentra después del signo # (para los enlaces de las anclas) http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion hash = #seccion
host	El nombre del servidor http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion host = www.ejemplo.com
hostname	La mayoría de las veces coincide con host, aunque en ocasiones, se eliminan las www del principio http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion hostname = www.ejemplo.com
href	La URL completa de la página actual http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion URL = http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion
pathname	Todo el contenido que se encuentra después del host http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion pathname = /ruta1/ruta2/pagina.html
port	Si se especifica en la URL, el puerto accedido http://www.ejemplo.com:8080/ruta1/ruta2/pagina.html#seccion port = 8080 La mayoría de URL no proporcionan un puerto, por lo que su contenido es vacío http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion port = (vacío)
protocol	El protocolo empleado por la URL, es decir, todo lo que se encuentra

Propiedad	Descripción
	antes de las dos barras inclinadas // http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion protocol = http:
search	Todo el contenido que se encuentra tras el símbolo ?, es decir, la consulta o *"query string"* http://www.ejemplo.com/pagina.php?variable1=valor1&variable2=valor2 search = ?variable1=valor1&variable2=valor2

De todas las propiedades, la más utilizada es **location.href**, que permite obtener o establecer la dirección de la página que se muestra en la ventana del navegador.

Además de las propiedades de la tabla anterior, el objeto location contiene numerosos métodos y funciones. Algunos de los métodos más útiles son los siguientes:

// Método assign()

location.assign("http://www.ejemplo.com"); // Equivalente a location.href = "http://www.ejemplo.com"

// Método replace()

location.replace("http://www.ejemplo.com");

// Similar a assign(), salvo que se borra la página actual del array history del navegador

// Método reload()

location.reload(true);

/* Recarga la página. Si el argumento es true, se carga la página desde el servidor. Si es false, se carga desde la cache del navegador */

6. El objeto navigator:

El objeto **navigator** es uno de los primeros objetos que incluyó el BOM y permite obtener información sobre el propio navegador. En Internet Explorer, el objeto navigator también se puede acceder a través del objeto clientInformation.

El objeto **navigator** se emplea habitualmente para detectar el tipo y/o versión del navegador en las aplicaciones cuyo código difiere para cada navegador. Además, se

emplea para detectar si el navegador tiene habilitadas las cookies y Java y también para comprobar los plugins disponibles en el navegador.

7. El 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.

Las siguientes propiedades están disponibles en el objeto screen:

Propiedad	Descripción
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 cuanto se puede redimensionar una ventana y para colocar una ventana centrada en la pantalla del usuario.

El siguiente ejemplo redimensiona una nueva ventana al tamaño máximo posible según la pantalla del usuario:

```
window.moveTo(0, 0);

window.resizeTo(screen.availWidth, screen.availHeight);
```

El siguiente ejemplo muestra algunas de las opciones de manejo de BOM

```
//Manejo de BOM (Browser Object Model)

function getBom(){
    console.log(window.innerHeight);
```

```
        console.log(window.innerWidth);
        console.log(window.location);
    }
function redirec(url){
    window.location.href=url;
}
getBom();
    redirec("http://www.google.es");
```

8. Eventos asociados a la carga de la pagina: DOMContentLoaded, load, beforeunload, unload

El ciclo de vida de una página HTML tiene tres eventos importantes:

- **DOMContentLoaded:** El navegador HTML está completamente cargado y el árbol DOM está construido, pero es posible que los recursos externos como <img> y hojas de estilo aún no se hayan cargado.
- **load:** No solo se cargó el HTML, sino también todos los recursos externos: imágenes, estilos, etc.
- **beforeunload/unload** : El usuario sale de la pagina.

Cada evento puede ser útil:

- **Evento DOMContentLoaded** – DOM está listo, por lo que el controlador puede buscar nodos DOM, inicializar la interfaz.
- **Evento load** – se cargan recursos externos, por lo que se aplican estilos, se conocen tamaños de imagen, etc.
- **Evento beforeunload** – el usuario se va: podemos comprobar si el usuario guardó los cambios y preguntarle si realmente quiere irse.
- **Evento unload** – el usuario casi se fue, pero aún podemos iniciar algunas operaciones, como enviar estadísticas.

Exploremos los detalles de estos eventos.

- **DOMContentLoaded:** El evento DOMContentLoaded ocurre en el objeto document. Debemos usar addEventListener para capturarlo:

```
document.addEventListener("DOMContentLoaded", ready);
// no "document.onDOMContentLoaded = ..."
```

Por ejemplo:

```
*<script>
  function ready() {
    alert('DOM is ready');

    // la imagen aún no está cargada (a menos que se haya almacenado en caché), por lo que el tamaño es 0x0
    alert(`Image size: ${img.offsetWidth}x${img.offsetHeight}`);
  }

  document.addEventListener("DOMContentLoaded", ready);
```

```
</script>

<img id="img" src="coche.jpg">
```

En el ejemplo, el controlador del evento DOMContentLoaded se ejecuta cuando el documento está cargado, por lo que puede ver todos los elementos, incluido el <img> que está después de él. Pero no espera a que se cargue la imagen. Entonces, alert muestra los tamaños en cero.

A primera vista, el evento **DOMContentLoaded** es muy simple. El árbol DOM está listo – aquí está el evento. Sin embargo, hay algunas peculiaridades.

- **DOMContentLoaded y scripts:** Cuando el navegador procesa un documento HTML y se encuentra con una etiqueta <script>, debe ejecutarla antes de continuar construyendo el DOM. Esa es una precaución, ya que los scripts pueden querer modificar el DOM, e incluso hacer document.write en él, por lo que **DOMContentLoaded** tiene que esperar. Entonces DOMContentLoaded siempre ocurre después de tales scripts:

```
<script>
  document.addEventListener("DOMContentLoaded", () => {
    alert("DOM listo!");
  });
</script>

<script src="prueba.js"></script>

<script>
  alert("Libreria cargada, linea de script ejecutada");
</script>
```

En el ejemplo anterior, primero vemos “Biblioteca cargada …” y luego “¡DOM listo!” (se ejecutan todos los scripts).

- **DOMContentLoaded y estilos:** Las hojas de estilo externas no afectan a DOM, por lo que DOMContentLoaded no las espera. Pero hay una trampa. Si tenemos un script después del estilo, entonces ese script debe esperar hasta que se cargue la hoja de estilo:

```
<link type="text/css" rel="stylesheet" href="style.css">
<script>
  // el script no se ejecuta hasta que se cargue la hoja de estilo
  alert(getComputedStyle(document.body).marginTop);
</script>
```

La razón de esto es que el script puede querer obtener coordenadas y otras propiedades de elementos dependientes del estilo, como en el ejemplo anterior. Naturalmente, tiene que esperar a que se carguen los estilos. Como DOMContentLoaded espera los scripts, ahora también espera a los estilos que están antes que ellos.

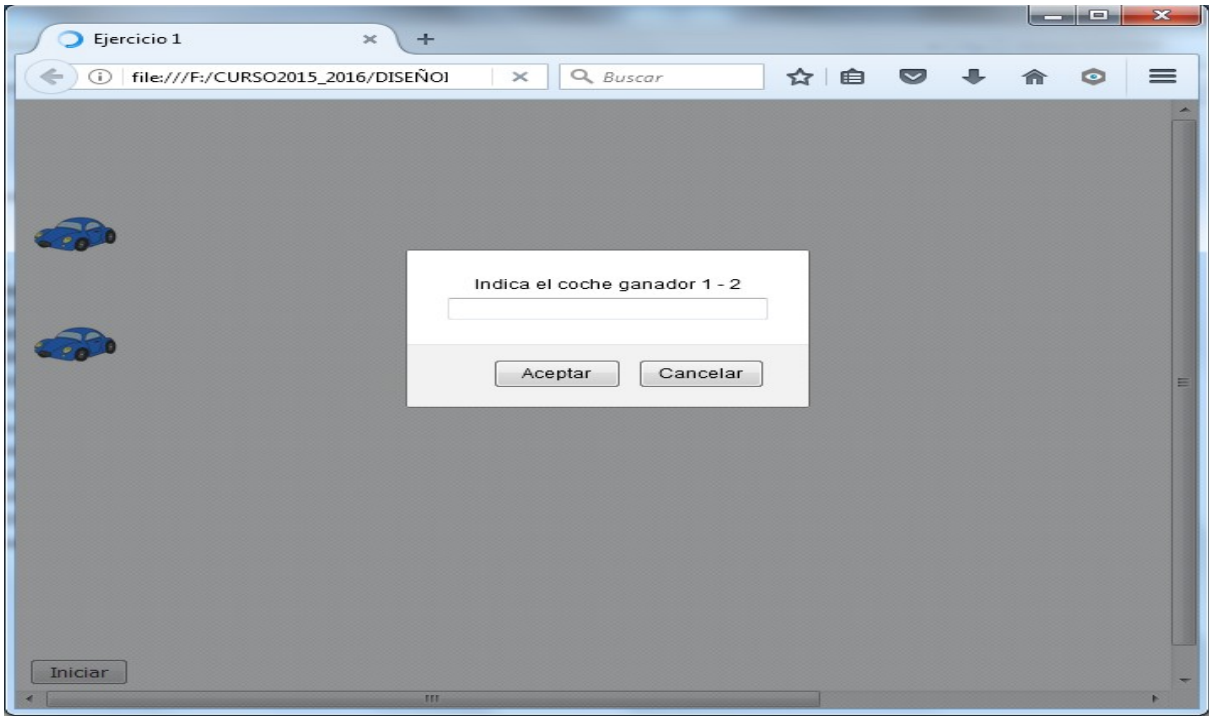
Secuencia y desarrollo:

1. **Ejercicio:** Implementa la siguiente página web en la que dos coches realizan una carrera. El usuario elige el coche que piensa que gana y lo guarda en una variable. A continuación se pulsa el botón de inicio y empiezan a moverse. La cantidad que hace que un coche se mueva más que otro se obtiene con la función **random**. Si el coche que llega antes es el que elige el usuario sale una alerta "Ha Ganado".

   FUNCIÓN RANDOM: Math.Random → Devuelve un número pseudoaleatorio de punto flotante entre 0 (inclusive) y 1 (exclusivo)

   OBTENER UN NUMERO ALEATORIO ENTRE DOS VALORES:

```
function getRandomArbitrary(min, max) {
        return Math.random() * (max - min) + min;}
```



   Requisitos:

   - Debes comprobar si los estilos han sido inicializados antes de comenzar la carrera.
   - Utilizar los métodos :
     - ***Element.setAttribute***: Establece el valor de un atributo en el elemento indicado. Si el atributo ya existe, el valor es actualizado, en caso contrario, el nuevo atributo es añadido con el nombre y valor indicado.
     - Para obtener el valor actual de un atributo, se utiliza ***getAttribute()***;
     - Para eliminar un atributo, se llama a ***removeAttribute()***.