UD3

Utilización de los objetos predefinidos de JavaScript - BOM

Clasificación de Objetos Predefinidos en JavaScript

Objetos Primitivos

Como hemos visto en la unidad anterior, en JavaScript tenemos los datos de **tipos primitivos** los cuales se clasifican básicamente en 5 tipos: **String, Number, Bool, Undefined y Null**:

```
//DATOS PRIMITIVOS
let cadena = "Hola Daw";
let pi = 3.14;
let bool = true;  //false
let noDefinidos = undefined;
let nulo = null;
```

Objetos Nativos

Luego por otro lado podemos definir los **objetos nativos**, los cuales no dependen del tipo del navegador y son los siguientes básicamente:

String	Number	Boolean
Date	Math	RegExp
Array	Function	Object

Objetos De Alto Nivel

Y luego tenemos otros objetos llamados “de alto nivel” lo cuales sí dependen del tipo de navegador (algunos son soportados por unos navegadores y otros no) los cuales se clasifican en:

Window	Screen	Navigator
Location	History	Document

Introducción a BOM (Browser Object Model)

Cuando el cliente web (Firefox, Chrome, etc.) carga un documento HTML creará una serie de objetos (objetos de alto nivel) que están intrínsecamente ligados al contexto del navegador web.

Todos estos objetos están comprendidos en el **BOM** (**Browser Object Model**) , que puede traducirse por **Modelo de Objetos del Navegador**. En este modelo se incluyen objetos que representan los diferentes elementos del navegador como la ventana, el historial, la dirección web y, sobre todo, los documentos HTML que se están mostrando. Estos documentos presentan una complejidad importante y requieren de un sistema de objetos propio, el **DOM** (**Document Object Model** , que puede traducirse por Modelo de Objetos del Documento).

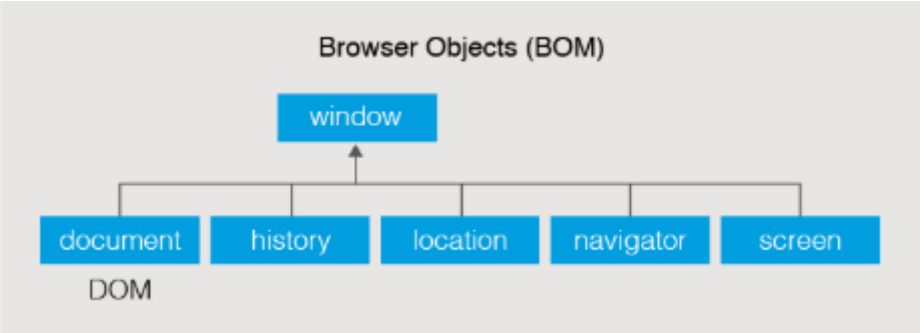
Mediante BOM, es posible redimensionar y mover la ventana del navegador, modificar el texto que se muestra en la barra de estado y realizar muchas otras manipulaciones no relacionadas con el contenido de la página HTML.

El mayor inconveniente de BOM es que, al contrario de lo que sucede con DOM, **ninguna entidad se encarga de "estandarizarlo"** o definir unos mínimos de interoperabilidad entre navegadores.

Algunos de los elementos que forman el **BOM** son los siguientes:

- 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.

El BOM está compuesto por varios objetos relacionados entre sí. El siguiente esquema muestra los objetos de BOM y su relación:



Importante: 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]    //Devuelve true
window.document === document    //Devuelve true
```

Veamos con un simple ejemplo algunos de los objetos mencionados anteriormente (necesitarás tener una imagen en la misma carpeta):

```
<html>
<head>
<meta http-equiv="content-type" content="text /html; charset=UTF-8">
<title> Objeto window </title>
</head>
<body>
<img src="./discoduro.jpg" name="img1"/>
<script>
console.log(window.document.img1.width);  // 0
console.log(document.img1.height);        // 0
console.log(window.location.href);        // url
console.log(location.protocol);           // http:
</script>
</body>
</html>
```

El objeto window

El objeto window **representa una ventana abierta 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.

Importante: Si en el HTML de una ventana contiene etiquetas de tipo **<iframe>** el navegador creará un objeto Window para la ventana principal y otros objetos Window para cada una de los <iframe>.

Para ver todas las propiedades y funciones que nos ofrece el objeto Window podemos dirigirnos al enlace de la W3Schools:

- El Objeto Window

Importante: No todos las **propiedades/métodos** están soportados por todos los navegadores. Para ello debemos de comprobar la compatibilidad. Una buena forma de hacer esto es dirigirnos al link anterior de la W3School y en cada propiedad/método se nos hace mención de su compatibilidad según con qué navegadores.

Propiedades de window

Las **propiedades de window** más relevantes son:

- **defaultStatus :** es el mensaje que se visualiza en la barra de estado del navegador.

- **frames :** es la matriz de la colección de frames que contiene una ventana. Por ejemplo, si queremos hacer referencia al primer frame de la ventana: **window.frames[0].frames.length** es el número de frames que contiene una ventana.
- **self :** hace referencia a la ventana actual.
- **top :** hace referencia a la raíz de la jerarquía de objetos.
- **window :** hace referencia a la ventana actual.
- **innerWidth , innerHeight :** anchura y altura interiores de la ventana.
- **closed :** devuelve un boolean indicando si la ventana se ha cerrado o no.
- **screenX , screenY :** coordenadas de la ventana en relación a la pantalla. Coordenadas del vértice superior izquierda donde está posicionada la ventana. Válido para los navegadores Google Chrome y IE (para Firefox utilizar **screenLeft** y **screenTop**).
- **opener :** devuelve una referencia de la ventana padre desde la que se ha creado la ventana hija.
- **localStorage :** devuelve una referencia al objeto localStorage que se utiliza para almacenar datos. Al contrario que las galletas, no tiene fecha de expiración.
- **sessionStorage :** devuelve un objeto storage para almacenar datos en una sesión web.

```
<html>
<head>
<meta http-equiv="content-type" content="text /html; charset=UTF-8">
<title> Objeto window </title>
</head>
<body>
<script>
console.log(window.innerWidth + '-' + window.innerHeight); // Puedes redimensionar
la ventana para ver cómo cambia este valor
console.log(window.screenX + '-' + window.screenY); // Si mueves la ventana de
posición para ver cómo cambia este valor
window.defaultStatus="Información en la barra de estado";
console.log(window.defaultStatus);
console.log(window.closed);     //false, porque la ventana está abierta.
</script>
</body>
</html>
```

Métodos de window

Los principales **métodos** del objeto window son:

- **alert (mensaje) :** crea una ventana de diálogo donde se muestra el mensaje.
- **confirm (mensaje) :** crea una ventana de confirmación (OK / Cancelar) donde se muestra el mensaje. Devuelve true o false .
- **prompt (mensaje, textoXDefecto) :** crea una ventana de diálogo donde se muestra el mensaje y permite la edición. El parámetro "texto" es el valor predeterminado. Al igual que confirm , contiene

Aceptar y **Cancelar**. Si aceptamos, el método devuelve el valor insertado (o el valor por defecto). Si cancela, devuelve **null** . Y siempre, todo lo que se introduzca se considera como un String.

- **close () :** sólo se pueden cerrar las ventanas que se han creado con un script.
- **stop () :** detiene la carga de la ventana.
- **setTimeout (expresión, msec) :** ejecuta la expresión que se pasa como argumento, una vez han pasado un número “msec” de milisegundos. Se utiliza para prorrogar la ejecución de la expresión. Cada vez que ejecutamos este método se devuelve un **id** para poder, si se quisiera, detenerlo.
- **clearTimeout (timeoutID) :** restaura la cuenta atrás iniciado por setTimeout .
- **setInterval (expresión, msec) :** ejecuta la expresión pasada como argumento cada **msec** milisegundos. Se utiliza para ejecutar la expresión a intervalos regulares de tiempo.
- **open (url, Window Name, params) :** crea una nueva ventana, le asocia el nombre WindowName , y accede a la URL que le hemos pasado. Le pasamos un conjunto de parámetros que describen las propiedades de la ventana . Entre otros: toolbar , width , height , left , top , fullscreen , resizable , location , status , Scrollbars , Menubar .

En el siguiente ejemplo utilizamos algunos de estos métodos:

```
<html>
<head>
<meta http-equiv="content-type" content="text /html; charset=UTF-8">
<title> Objeto window </title>
</head>
<body>
<script>
var myWindow;
var titulo;

myWindow = window.open ( "https://www.lingscars.com/", "NombreVentana", "width =
300, height = 300, left = 300, top = 300, resizable = 1");

setTimeout (function ()
{
      myWindow.close ();
      titulo=prompt ( "Pon el título de la ventana:", "Titulo");
}, 2000);

setTimeout (function ()
{
    crearVentana (titulo);
}, 6000);  // 2 segundos

function crearVentana(titulo)
{
      var opciones="toolbar=0, location=0, directories=0, status=0,";
      opciones+="Menubar=0, Scrollbars=0, resizable=0, copyhistory=0,";
      opciones+="width=100, height=100";
      window.open ( "", titulo, opciones);
}
```

```
</script>
</body>
</html>
```

También existen 4 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.
- **resizeBy(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.

A continuación se muestran algunos ejemplos de uso de estas funciones:

```
// Mover la ventana 20 píxel hacia la derecha y 30 píxel hacia abajo
window.moveBy(20, 30);   // Posición relativa
// Colocar la ventana en la esquina izquierda superior de la ventana
window.moveTo(0, 0);     // Posición Absoluta
// Redimensionar la ventana hasta un tamaño de 250 x 250
window.resizeTo(250, 250);
// Agrandar la altura de la ventana en 50 píxel
window.resizeBy(0, 50);
```

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() , setInterval() y clearTimeout (id) 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);
```

El método setTimeout() **requiere dos argumentos**. El primero es el código que se va a ejecutar o una referencia a la función que se debe ejecutar. El segundo argumento es el tiempo, en milisegundos, que se

espera hasta que comienza la ejecución del código. El ejemplo anterior se puede rehacer utilizando una función:

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

Como es habitual, **cuando se 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**.

Aquí tenemos un problema: ¿Y si quisiéramos pasar algún parámetro a la función? Tenemos dos opciones, bien creando variables de clase que puedan ser accesibles en el ámbito de la función, y así no vernos obligados a pasar ningún parámetro, y la segunda opción es pasar un solo argumento como tercer valor de la función setTimeout(): `setInterval(function, milliseconds, param1, param2, ...)`

```
setTimeout(muestraMensaje, 3000 , parametro1, parametro2);
```

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():

Veamos un ejemplo que engloba estas 3 métodos:

```
<html>
```

```
<head>
<meta http-equiv="content-type" content="text /html; charset=UTF-8">
<title> Objeto window </title>
</head>
<body>
<script>

function muestraMensaje() {
      console.log("Este mensaje se muestra cada segundo \n");
}
function seAcaboElMensaje(idIntervalo) {
      console.log("Parando de repetir el mensaje...");
      // Después de ejecutarse un determinado número de veces, se elimina el
intervalo
      clearInterval(idIntervalo);
      console.log("Hemos parado el intervalo con id: " + id);
}
var id = setInterval(muestraMensaje, 1000);
setTimeout(seAcaboElMensaje, 7000, id);

</script>
</body>
</html>
```

El objeto document NO ESTUDIAR EL “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

Las propiedades title y URL son de lectura y escritura, por lo que además de obtener su valor, se puede establecer de forma directa:

```
// modificar el título de la página
document.title = "Nuevo titulo";

// llevar al usuario a otra página diferente
document.URL = "http://nueva_pagina";
```

Además de propiedades, el objeto document contiene varios arrays con información sobre algunos elementos de la página:

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 párrafo de la página</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
```

Veamos un ejemplo de cómo podemos manejar/acceder a algunos de los elementos del documento HTML:

```
<html>
<head>
<meta http-equiv="content-type" content="text /html; charset=UTF-8">
<title> Objeto Document </title>
</head>
<body>
      <a href="https://www.lingscars.com/"> SuperWebSite </a><br/>
      <img src="./discoduro.jpg" name="img1"/>

<script>
console.log ( "Location:" + document.location);
console.log ( "Título:" + document.title);
console.log ( "Última Modificación:" + document.lastModified);

linkDocumento = document.links;
for(var i=0; i< linkDocumento.length; i++) {
  console.log("Link Documento Num " + i + ": " + linkDocumento[i].href);
  console.log("URL: "+ linkDocumento.item(0).href);
}

console.log (document.images[0].src);
</script>
</body>
</html>
```

El objeto location

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 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():** Reemplaza la url actual y navega hacia esa url
  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 caché del navegador */

En el siguiente ejemplo podemos ver en funcionamiento las propiedades y métodos explicados del objeto Location:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>Objeto Location. Propiedades y Métodos</title>
<script type="text/javascript">
      function assign_url () {
      document.location.assign('https://tenor.com/search/rajoy-gifs');
      }
      function reload_url () {
      document.location.reload();
      }
      function replace_url () {
      document.location.replace('http://www.meneame.net'); <!-- Fijaros que no
tenemos el histórico para ir atrás -->
      }
console.log('location.hash: ' + location.hash);
console.log('location.hostname: ' + location.hostname);
console.log('location.href: ' + location.href);
console.log('location.origin: ' + location.origin);
console.log('location.pathname: ' + location.pathname);
console.log('location.port: ' + location.port);
console.log('location.protocol: ' + location.protocol);
console.log('location.search: ' + location.search);
</script>
</head>
<body>
<a href="http://www.xataka.com">IOC</a><br />
<img src="http://ioc.xtec.cat/educacio/templates/ioc/images/ioc_logo.png" /img><br
/>
<button onclick="assign_url();">Método location.assign</button>
<button onclick="reload_url();">Método location.reload</button>
<button onclick="replace_url();">Método location.replace</button>
</body>
</html>
```

El objeto history

El objeto history **contiene las URLs visitadas por el usuario** (en el marco de una ventana del navegador). El objeto de histórico es parte del objeto window.

Propiedades de history

La propiedad más importante del objeto **history** es:

- **length :** devuelve el número de URLs en la lista del histórico de navegación.

Métodos de history

Los métodos más importantes del objeto **history** son:

- **back () :** carga la URL previa en la lista del histórico de navegación.
- **forward () :** carga la siguiente URL en la lista del histórico de navegación.
- **go (number numeroPaginas) :** carga una URL específica en la lista del histórico de navegación. El parámetro es el número de páginas del histórico que debe retroceder. **El número debe de ser en negativo.** Una vez que hayamos retrocedido en el "history" del navegador, se perderán esas referencias que hemos saltado.

Como ejemplo, y siempre dentro de la misma ventana, trata de de navegar por una serie de páginas webs siguientes URLs, y finalmente introducir la URL del código que se propone.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>Objecto history. Propiedades y métodos</title>
</head>
<body>
<button onclick="console.log('Número elements: ' + history.length);"> Número de
elementos de la lista. </button>
<button onclick="history.back()">URL anterior</button>
</body>
</html>
```

El objeto navigator

Permite obtener información sobre el propio navegador. En Internet Explorer, el objeto navigator también se puede acceder a través del objeto **clientInformation**.

Aunque es uno de los objetos menos estandarizados, algunas de sus propiedades son comunes en casi todos los navegadores. A continuación se muestran algunas de esas propiedades:

Propiedad	Descripción
appCodeName	Cadena que representa el nombre del navegador (normalmente es Mozilla si lo ejecutas en el Mozilla).
appName	Cadena que representa el nombre oficial del navegador

appMinorVersion	(Sólo Internet Explorer) Cadena que representa información extra sobre la versión del navegador
appVersion	Cadena que representa la versión del navegador.
browserLanguage	Cadena que representa el idioma del navegador.
cookieEnabled	Boolean que indica si las cookies están habilitadas
cpuClass	(Sólo Internet Explorer) Cadena que representa el tipo de CPU del usuario ("x86", "68K", "PPC", "Alpha", "Other")
javaEnabled	Boolean que indica si Java está habilitado
language	Cadena que representa el idioma del navegador
mimeTypes	Array de los tipos MIME registrados por el navegador
onLine	(Sólo Internet Explorer) Boolean que indica si el navegador está conectado a Internet
oscpu	(Sólo Firefox) Cadena que representa el sistema operativo o la CPU
platform	Cadena que representa la plataforma sobre la que se ejecuta el navegador
plugins	Array con la lista de plugins instalados en el navegador
preference()	(Sólo Firefox) Método empleado para establecer preferencias en el navegador
product	Cadena que representa el nombre del producto (normalmente, es Gecko en Chrome)
productSub	Cadena que representa información adicional sobre el producto (normalmente, la versión del motor Gecko)
securityPolicy	Sólo Firefox
systemLanguage	(Sólo Internet Explorer) Cadena que representa el idioma del sistema operativo
userAgent	Cadena que representa la cadena que el navegador emplea para identificarse en los servidores

userLanguage	(Sólo Explorer) Cadena que representa el idioma del sistema operativo
userProfile	(Sólo Explorer) Objeto que permite acceder al perfil del usuario

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.

Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>Objecto Navigator</title>
</head>
<body>
<script>
console.log("appCodeName: " + navigator.appCodeName);
console.log("appName: " + navigator.appName);
console.log("appVersion: " + navigator.appVersion);
console.log("cookieEnabled: " + navigator.cookieEnabled);
console.log("language: " + navigator.language);
console.log("onLine: " + navigator.onLine);
console.log("platform: " + navigator.platform);
console.log("product: " + navigator.product);
console.log("userAgent: " + navigator.userAgent);
</script>
</body>
</html>
```

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. **Este objeto no tiene métodos, solo propiedades**.

→ **Información:** A los diseñadores web siempre les ha preocupado saber cuál es la resolución del navegador del cliente para que las páginas web se vean bien en los diferentes formatos de pantalla. Hoy en día, gracias a frameworks como bootstrap , la adaptación de la web en diferentes formatos de pantalla es mucho más fácil.

Las siguientes propiedades están disponibles en el objeto screen:

Propiedad	Descripción
availHeight	Altura de pantalla disponible para las ventanas. En otras palabras, la altura máxima a la que se puede maximizar la ventana del navegador.
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. Coincide con la resolución de la pantalla.
width	Anchura total de la pantalla en píxel. Coincide con la resolución de la pantalla.

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.

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);
```