UNIDAD DIDACTICA 04

MODELO DE OBJETOS

PREDEFINIDOS EN JAVASCRIPT.

Índice

Objetivos.

- ✓ Conocer cuáles son los principales objetos predefinidos de JavaScript.
- ✓ Comprender las propiedades y métodos de los principales objetos de JavaScript.
- ✓ Aprender a generar código HTML desde sentencias JavaScript.
- ✓ Dominar el uso de los marcos de HTML y aprender a realizar interacciones entre ellas con JavaScript.
- ✓ Manipular y gestionar la creación y apariencia de las ventanas del navegador además de la comunicación entre ellas.

1.- Introducción.

Una página web, es un documento HTML que será interpretado por los navegadores de forma gráfica, pero que también va a permitir el acceso al código fuente de la misma.

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.

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.

Ahora que ya hemos visto los fundamentos de la programación, vamos a profundizar un poco más en lo que se refiere a los objetos, que podremos colocar en la mayoría de nuestros documentos.

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

Las ***propiedades*** son como los adjetivos en el lenguaje, es decir que expresan una cualidad del objeto, así por ejemplo las instrucciones dadas en lenguaje CSS se consideran propiedades, ya que modifican el aspecto de la página. Otras propiedades pueden darnos información acerca del objeto, por ejemplo, sobre su tamaño, valor, estado, etc. Los atributos de las etiquetas HTML son considerados también propiedades.

Los ***métodos*** son funcionalidades, es decir funciones o modos de operar asociados a un objeto. Se comportan igual que una función. Un método es algo que se puede hacer con el objeto. Es por eso que llevan siempre un paréntesis detrás, en

el que se le pueden pasar varios valores para que opere con ellos (los parámetros o argumentos).

La forma de acceder a una propiedad de un objeto es la siguiente:

```
nombreobjeto.propiedad
```

La forma de acceder a un método de un objeto es la siguiente:

```
nombreobjeto.metodo( [parámetros opcionales] )
```

Ejemplo:

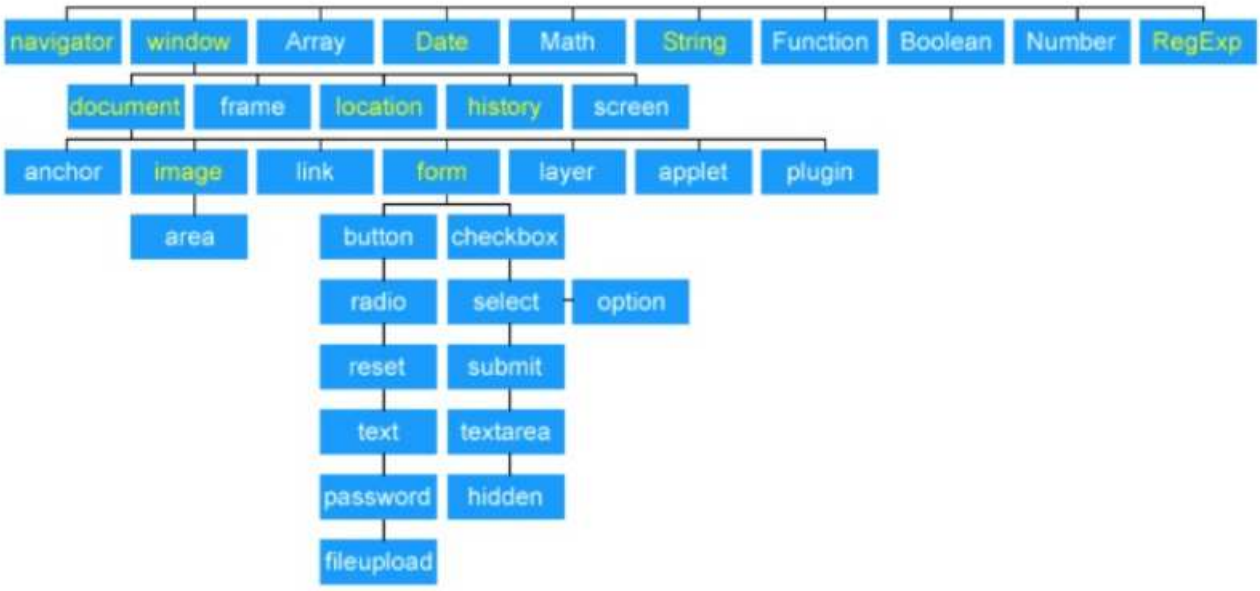
```
Math.SQRT2
fecha.getFullYear()
```

También podemos referenciar a una propiedad de un objeto, por su índice en la creación. Los índices comienzan por 0.

En esta unidad, nos enfocaremos en objetos de alto nivel, que encontraremos frecuentemente en nuestras aplicaciones de JavaScript: **`window`**, **`location`**, **`navigator`** y **`document`**. El objetivo, no es solamente indicarte las nociones básicas para que puedas comenzar a realizar tareas sencillas, sino también, el prepararte para profundizar en las propiedades y métodos, gestores de eventos, etc. Que encontraremos en unidades posteriores.

En esta unidad, vamos a ver solamente las propiedades básicas, y los métodos de los objetos mencionados anteriormente.

Los objetos en JavaScript se ordenan de modo jerárquico, es muy importante que tengamos este gráfico en la mente, sobre todo los objetos en amarillo



2.-Objetos de más alto nivel en JavaScript.

2.1.- Objeto window.

En la jerarquía de objetos, tenemos en la parte superior el objeto **`window`**.

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

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

Cómo se ve en el gráfico de la jerarquía de objetos, debajo del objeto **`window`** tenemos otros objetos como el **`navigator`**, **`screen`**, **`history`**, **`location`** y el objeto **`document`**. Este objeto **`document`** será el que contendrá toda la jerarquía de objetos, que tengamos dentro de nuestra página HTML.

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

Acceso a propiedades y métodos.

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

```
window.nombrePropiedad
window.nombreMétodo( [parámetros] )
```

Como puedes ver, los parámetros van entre corchetes, indicando que son opcionales y que dependerán del método al que estemos llamando.

Un objeto **`window`** también se podrá referenciar mediante la palabra **`self`**, cuando estamos haciendo la referencia desde el propio documento contenido en esa ventana:

```
self.nombrePropiedad
self.nombreMétodo( [parámetros] )
```

Podremos usar cualquiera de las dos referencias anteriores, pero intentaremos dejar la palabra reservada `self`, para scripts más complejos en los que tengamos múltiples marcos y ventanas.

Debido a que el objeto `window` siempre estará presente cuando ejecutemos nuestros scripts, podremos omitirlo, en referencias a los objetos dentro de esa ventana. Así que, si escribimos:

```
nombrePropiedad
nombreMétodo( [parámetros] )
```

También funcionaría sin ningún problema, porque se asume que esas propiedades o métodos, son del objeto de mayor jerarquía (el objeto `window`) en el cual nos encontramos.

2.1.1- Gestión de ventanas.

Un script no creará nunca la ventana principal de un navegador. Es el usuario, quien realiza esa tarea abriendo una URL en el navegador o un archivo desde el menú de abrir. Pero, sin embargo, un script que esté ejecutándose en una de las ventanas principales del navegador, sí que podrá crear o abrir nuevas sub-ventanas.

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

Por ejemplo, si consideramos la siguiente instrucción que abre una nueva ventana de un tamaño determinado y con el contenido de un documento HTML:

```
var
subVentana=window.open("nueva.html","nueva","height=800,width=600");
```

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

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

Ejemplo de cómo abrir y cerrar una sub-ventana:

```
<!DOCTYPE html>
<html>
  <head>
  <meta http-equiv="content-type" content="text/html;charset=utf-8">
  <title>Apertura y Cierre de Ventanas</title>
  <script type="text/javascript">
```

```
function inicializar()
{
   document.getElementById("crear-ventana").onclick=crearNueva;
   document.getElementById("cerrar-ventana").onclick=cerrarNueva;
}
var nuevaVentana;
function crearNueva()
{
   nuevaVentana =
   window.open("http://www.google.es","","height=400,width=800");
}
function cerrarNueva()
{
   if (nuevaVentana){
   nuevaVentana.close(); nuevaVentana = null;
}
}
</script>
</head>
<body onLoad="inicializar()">
   <h1>Abrimos y cerramos ventanas</h1>
   <form>
      <p> <input type="button" id="crear-ventana" value="Crear Nueva
      Ventana">
      <input type="button" id="cerrar-ventana" value="Cerrar Nueva
      Ventana"> </p>
   </form>
</body>
</html>
```

2.1.2- Propiedades y métodos.

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

Propiedades del objeto window	
Propiedad	**Descripción**
`closed`	Devuelve un valor `Boolean` indicando cuando una ventana ha sido cerrada o no.
`defaultStatus`	Ajusta o devuelve el valor por defecto de la barra de estado de una ventana.
`document`	Devuelve el objeto `document` para la ventana.
`frames`	Devuelve un array de todos los marcos (incluidos `iframes`) de la ventana actual.
`history`	Devuelve el objeto `history` de la ventana.
`innerHeight`	Corresponde a la altura utilizable de la ventana.

`innerWidth`	Corresponde al ancho utilizable de la ventana.
`length`	Devuelve el número de `frames` (incluyendo `iframes`) que hay dentro de una ventana.
`location`	Devuelve la localización del objeto ventana (URL del fichero)
`locationBar`	Corresponde a la barra de direcciones del navegador.
`menuBar`	Corresponde a la barra de menú del navegador.
`name`	Ajusta o devuelve el nombre de una ventana.
`navigator`	Devuelve el objeto `navigator` de una ventana.
`opener`	Devuelve la referencia a la ventana que abrió la ventana actual.
`outerHeight`	Corresponde a la altura exterior de la página.
`outerWidth`	Corresponde al ancho exterior de la página.
`pageXoffset`	Corresponde a la posición horizontal de la ventana..
`pageYoffset`	Corresponde a la posición vertical de la ventana
`parent`	Devuelve la ventana padre de la ventana actual.
`personalBar`	Corresponde a la barra de herramientas personal.
`scrollbars`	Corresponde a las barras de desplazamiento horizontal y vertical.
`self`	Devuelve la ventana actual.
`status`	Ajusta el texto de la barra de estado de una ventana.
`toolbar`	Corresponde a la barra de herramientas del navegador.
`top`	Corresponde a la ventana de nivel superior.

Todas estas propiedades podemos manipularlas con el fin de mostrar o no una parte de la ventana del navegador o modificar algunas características como su tamaño o posición.

Métodos del objeto window	
Métodos	**Descripción**
`alert()`	Muestra una ventana emergente de alerta y un botón de aceptar.
`blur()`	Elimina e foco de la ventana actual.
`close()`	Cierra la ventana actual.
`confirm()`	Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.
`find()`	Muestra una ventanita de búsqueda.
`focus()`	Coloca el foco en la ventana actual.
`forward()`	Ir una página adelante en el historial de páginas visitadas. Como si pulsásemos el botón de adelante del navegador.
`home()`	Ir a la página de inicio que haya configurada en el explorador.
`moveBy(pixelsX, pixelsY)`	Mueve la ventana del navegador los pixels que se indican por parámetro hacia la derecha y abajo.
`moveTo(pixelsX, pixelsY)`	Mueve la ventana del navegador a la posición indicada en las coordenadas que recibe por parámetro.
`open()`	Abre una nueva ventana de navegación.
`print()`	Abre una ventana secundaria del navegador.
`prompt()`	Muestra una ventana de dialogo para introducir datos.
`resizeBy(pixelsAncho, pixelsAlto)`	Redimensiona el tamaño de la ventana, añadiendo a su tamaño actual los valores indicados en los parámetros. El primero para la altura y el segundo para la anchura. Admite valores negativos si se desea reducir la ventana.
`resizeTo(pixelsAncho, pixelsAlto)`	Redimensiona la ventana del navegador para que ocupe el espacio en pixels que se indica por parámetro.
`scrollBy(pixelsX, pixelsY)`	Hace un scroll del contenido de la ventana relativo a la posición actual.
`scrollTo(pixelsX, pixelsY)`	Hace un scroll de la ventana a la posición indicada por el parámetro. Este método se tiene que utilizar en lugar de scroll.
`stop()`	Detiene una página.

Ejemplos:

Caja de alerta:

```
window.alert("Este es el texto que sale")
```

es lo mismo que poner:

```
alert("Este es el texto que sale")
```

Caja de confirmación:

```
<script>
   var respuesta = confirm("Aceptame o rechazame")
   alert ("Has pulsado: " + respuesta)
</script>
```

Caja de introducción de un dato:

```
<script>
   nombre = null
   while (nombre == null || nombre == "")
   {
      nombre = prompt("Dime tu nombre:","")
   }
   document.write("<h1>Hola " + nombre + "</h1>")
</script>
```

2.2.- Objeto location.

El objeto `location` contiene información referente a la URL actual.

Este objeto, es parte del objeto `window` y accedemos a él a través de la propiedad `window.location`.

El objeto `location` contiene información referente a la URL actual.

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

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

Ejemplo:

```
<script type=”text/javascript”>
   Document.write(“URL Competa: “ + location.href + “<br>”);
   Document.write(“Pathname: “ + location.pathname+ “<br>”);
   Document.write(“Protocolos:“ +location.protocol+ “<br>”);
</script>
<form>
   <input type="Button" value="Recargar"
   onclick="location.reload();">
</form>
```

2.3.- Objeto navigator.

Este objeto `navigator`, contiene información sobre el navegador que estamos utilizando cuando abrimos una URL o un documento local.

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

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

2.4.- Objeto document.

Cada documento cargado en una ventana del navegador, será un objeto de tipo `document`.

El objeto `document` proporciona a los scripts, el acceso a todos los elementos HTML dentro de una página.

Este objeto forma parte además del objeto `window`, y puede ser accedido a través de la propiedad `window.document` o directamente `document` (ya que podemos omitir la referencia a la `window` actual).

Colección del objeto document	
Métodos	**Descripción**
`anchor[]`	Es un array que contiene todos los hiperenlaces del documento.
`applets[]`	Es un array que contiene todos los applets del documento.
`forms[]`	Es un array que contiene todos los formularios del documento.
`images[]`	Es un array que contiene todos las imágenes del documento
`links[]`	Es un array que contiene todos los enlaces del documento.

Propiedades del objeto document	
Propiedad	**Descripción**
`cookie`	Devuelve todos los nombres/valores de las cookies en el documento.
`domain`	Cadena que contiene el nombre de dominio del servidor que cargo el documento.
`lastModified`	Devuelve la fecha y hora de la última modificación del documento.
`layers`	Corresponde a las capas (etiqueta `<layer>`) del documento.
`readyState`	Devuelve el estado de carga del documento.
`referrer`	Cadena que contiene la URL del documento desde el cual llegamos al documento actual.
`title`	Devuelve o ajusta el título del documento.
`URL`	Devuelve la URL completa del documento.

Métodos del objeto document	
Métodos	**Descripción**
`captureEvent()`	Intercepta un evento para que pueda ser manipulado por el documento.
`close()`	Cierra el flujo abierto previamente con el documento open().
`getElementById()`	Para acceder a un elemento identificado por el id escrito entre paréntesis.
`getElementByName()`	Para acceder a los elementos identificados por el atributo `name` escrito entre paréntesis.
`getElementByTagName()`	Para acceder a los elementos identificados por el `tag` o la etiqueta escrita entre paréntesis.
`getSelection()`	Devuelve el texto seleccionado en el documento.
`home()`	Carga la página de inicio,
`open()`	Abre el flujo de escritura para poder utilizar `documnet.write()` o `docuemnt.writeln` en el documento.
`write()`	Para poder escribir expresiones HTML o código JavaScript dentro de un documento.
`writeln()`	Lo mismo que `write()` pero añade un salto de línea al final de cada instrucción.

Ejemplo:

```
<script>
   function cambiaColor(colorin)
   {
     document.bgColor = colorin
   }
</script>
<form>
```

```
   <input type="Button" value="Rojo"
     onclick="cambiaColor('ff0000')">
   <input type="Button" value="Verde"
     onclick="cambiaColor('00ff00')">
   <input type="Button" value="Azul"
     onclick="cambiaColor('0000ff')">
</form>
```

2.5.- Objeto history.

Este objeto almacena las referencias de todos los sitios web visitados. Estas referencias se guardan en una lista y sus propiedades y métodos se utilizan principalmente para que el usuario de una aplicación web pueda desplazarse para adelante y para atrás. Sin embargo, al ser una lista de referencia, no podemos acceder a los nombres de las direcciones URL, visitadas, ya que son información del usuario.

Propiedades del objeto history	
Propiedad	**Descripción**
`currennt`	Corresponde a la cadena que contiene la URL de la entrada actual del historial.
`length`	Corresponde al número de páginas que han sido visitadas.
`next`	Corresponde a la cadena que contiene la siguiente entrada del historial.
`previous`	Corresponde a la cadena que contiene la anterior entrada del historial.

Métodos del objeto history	
Métodos	**Descripción**
`back()`	Carga la URL del documento anterior del historial.
`forward()`	Carga la URL del documento siguiente del historial.
`go()`	Carga la URL del documento especificado por el índice que pasamos como parámetro dentro del historial.

3.- Marcos.

La mayor parte de las aplicaciones web se crean para ser ejecutadas en una sola ventana, aunque exista la posibilidad de dividir la ventana en dos o más partes independientes. En estos sectores podemos utilizar JavaScript para tener una interacción entre ellos. Estos sectores se denominan marcos. Algunas páginas web presentan una estructura en la cual una parte de la página permanece fija mientras que otra parte va cambiando en base a la navegación que se efectúa en otro marco. En realidad, este efecto se produce creando diferentes páginas web y posicionando cada una de ellas en un marco diferente.

Un objeto `frame`, representa un marco HTML. La etiqueta `<frame>` identifica una ventana particular, dentro de un conjunto de marcos (`frameset`).

Todo lo anterior se aplicará también al objeto `iframe <iframe>.`

Los marcos se definen utilizando HTML. Estos presentan la ventaja de poder crear páginas en las que es posible mantener constantemente algunos elementos como botones, enlaces, imágenes, etc. con JavaScript podemos manipular los marcos y realizar una interacción entre ellos. De este modo es posible utilizar todas las ventajas del lenguaje sobre los marcos.

Para utilizar correctamente los marcos de HTML, necesitamos conocer un par de etiquetas con sus respectivos atributos:

- `<frameset>:` las funciones básicas de esta etiqueta son indicar al navegador que la página contiene marcos, definir el número de marcos que se utilizarán en la página y establecer el tamaño de cada marco. Los principales atributos son `cols` y `rows`, los cuales dividen respectivamente el conjunto de marcos vertical u horizontalmente.
- `<frame>`: esta etiqueta define las características de cada marco. Para cada etiqueta `<frame>` en un documento HTML, se creará un objeto `frame.`

Propiedades del objeto frame/Iframe	
Propiedad	**Descripción**
`align`	Cadena que contiene el valor del atributo `align` (alineación) en un `iframe`.
`contentDocument`	Devuelve el objeto documento contenido en u `frame/iframe`.
`contentWindow`	Devuelve el objeto `window` generado por un `frame/iframe`.
`frameBorder`	Cadena que contiene el valor del atributo `frameborder` (borde del marco) de un `frame/iframe`.
`height`	Cadena que contiene el valor del atributo `height` (altura) de un `iframe`.
`longDesc`	Cadena que contiene el valor del atributo `longdesc` (descripción larga) de un `frame/iframe`.
`marginHeight`	Cadena que contiene el valor del atributo `marginheight` (alto del margen) de un `frame/iframe`.
`marginWidth`	Cadena que contiene el valor del atributo `marginwidth` (ancho del margen) de un `frame/iframe.`
`name`	Cadena que contiene el valor del atributo `name` (nombre) de un `frame/iframe.`
`noResize`	Cadena que contiene el valor del atributo `noresize` de un `frame/iframe.`
`scrolling`	Cadena que contiene el valor del atributo `scrolling` (desplazamiento) de un `frame/iframe.`
`src`	Cadena que contiene el valor del atributo `src` (origen) de un `frame/iframe.`
`width`	Cadena que contiene el valor del atributo `width` (ancho) de un `frame/iframe`

Eventos del objeto frame/iframe	
Eventos	**Descripción**
`onload`	Script que se ejecutara inmediatamente después a que se cargue el `frame/frame`..

3.1.- Jerarquías.

Uno de los aspectos más atractivos de JavaScript en aplicaciones cliente, es que permite interacciones del usuario en un marco o ventana, que provocarán actuaciones en otros marcos o ventanas. En esta sección vamos a ver algunas nociones para trabajar con múltiples ventanas y marcos.

Marcos: Padres e Hijos.

En el gráfico de jerarquías de objetos, vimos como el objeto `window` está en la cabeza de la jerarquía y puede tener sinónimos como `self`. En esta sección veremos que, cuando trabajamos con marcos o `iframes`, podemos referenciar a las ventanas como: `frame`, `top` y `parent`.

Aunque el uso de marcos o `iframes` es completamente válido en HTML, en términos de usabilidad y accesibilidad no se recomiendan, por lo que su uso está en verdadero declive. El problema fundamental con los marcos, es que las páginas contenidas en esos marcos no son directamente accesibles, en el sentido de que, si navegamos dentro de los `frames`, la URL principal de nuestro navegador no cambia, con lo que no tenemos una referencia directa de la página en la que nos encontramos. Esto incluso es mucho peor si estamos accediendo con dispositivos móviles. Otro problema con los `frames` es que los buscadores como Google, Bing, etc., no indexan bien los `frames`, en el sentido de que si por ejemplo registran el contenido de un `frame`, cuando busquemos ese contenido, nos conectará directamente con ese `frame` como si fuera la página principal, con lo que la mayoría de las veces no tenemos referencia de la sección del portal o web en la que nos encontramos.

Ejemplo de Frame:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<html>
   <head>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-
      8">
     <title>Titulo para todas las páginas de este conjunto de
       Frames</title>
   </head>
     <frameset cols="50%,50%">
     <frame name="frameIzdo" src="documento1.html" title="Frame 1">
     <frame name="frameDcho" src="documento2.html" title="Frame 2">
     <noframes></noframes>
    </frameset>
</html>
```

Este código divide la ventana del navegador en dos marcos de igual tamaño, con dos documentos diferentes en cada columna. Un `frameset` establece las relaciones entre los marcos de la colección. El `frameset` se cargará en la ventana principal (ventana padre), y cada uno de los marcos (`frames`) definidos dentro del `frameset`, será un marco hijo (ventanas hijas).

En la siguiente figura se puede observar la jerarquía resultante:

Fíjate en el gráfico que la ventana padre (la que contiene el `frameset`), no tiene ningún objeto `document` (ya que el `frameset` no puede contener los objetos típicos del HTML como formularios, controles, etc.) y son los `frames` hijos, los que sí tienen objeto `document`. El objeto `document` de un marco, es independiente del objeto `document` del otro marco, y en realidad cada uno de los marcos, será un objeto `window` independiente.

3.2.- Comunicación entre marcos.

Referencias Padres- Hijos.

Desde el momento en el que el documento padre contiene uno o más marcos, ese documento padre mantiene un array con sus marcos hijo. Podemos acceder a un marco a través de la sintaxis de array o por el nombre que le hemos dado a ese marco, por el id o con el atributo `name` que hemos puesto en la marca `<frame>`.

Ejemplos de referencias a los marcos hijo:

(Recordar que todo lo que va entre corchetes [] es opcional).

```
[window.]frames[n].objeto-función-variable-nombre
[window.]frames["nombreDelMarco"].objeto-función-variable-nombre
[window.]nombreDelMarco.objeto-función-variable-nombre
```

El índice numérico n, que indica el número de `frame`, está basado en el orden en el que aparecen en el documento `frameset`. Se recomienda que pongamos un nombre a cada `frame` en dicho documento, ya que así la referencia a utilizar será mucho más fácil.

Referencias Hijo-Padre.

Es bastante más común enlazar scripts al documento padre (`frameset`), ya que éste se carga una vez y permanecerá cargado con los mismos datos, aunque hagamos modificaciones dentro de los marcos.

Desde el punto de vista de un documento hijo (aquel que está en un `frame`), su antecesor en la jerarquía será denominado el padre (`parent`). Por lo tanto, para hacer referencia a elementos del padre se hará:

```
parent.objeto-función-variable-nombre
```

Si el elemento al que accedemos en el padre es una función que devuelve un valor, el valor devuelto será enviado al hijo sin ningún tipo de problemas. Por ejemplo:

```
var valor=parent.NombreFuncion();
```

Además, como la ventana padre está en el top de la jerarquía de ventanas, opcionalmente podríamos escribir:

```
var valor=top.NombreFuncion();
```

Referencias Hijos-Hijos.

El navegador necesita un poco más de asistencia cuando queremos que una ventana hija se comunique con una hermana. Una de las propiedades de cualquier ventana o marco es su padre (`parent` – el cuál será `null` cuando estamos hablando de una ventana sin hijos). Por lo tanto, la forma de comunicar dos ventanas o marcos hermanos va a ser siempre referenciándolos a través de su padre, ya que es el único nexo de unión entre ambos (los dos tienen el mismo padre).

Podemos utilizar alguno de los siguientes formatos:

```
parent.frames[n].objeto-función-variable-nombre
parent.frames["nombreDelMarco"].objeto-función-variable-nombre
parent.nombreDelMarco.objeto-función-variable-nombre
```

3.3.- Comunicación entre múltiples ventanas.

En esta sección, vamos a ver cómo podemos comunicarnos con sub-ventanas, que abrimos empleando el método `open()` del objeto `window`.

Cada objeto `window` tiene una propiedad llamada `opener`. Esta propiedad contiene la referencia a la ventana o marco, que ha abierto ese objeto `window` empleando el método `open()`. Para la ventana principal el valor de `opener` será `null`.

Debido a que `opener` es una referencia válida a la ventana padre que abrió las otras, podemos emplearlo para iniciar la referencia a objetos de la ventana original (padre) desde la ventana hija. Es semejante a lo que vimos con `frames`, pero en este caso es entre ventanas independientes del navegador.

Ejemplo2.html

```
<html>
  <head>
```

```
    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <title>Ventana Principal</title>
    <script type="text/javascript">
     function abrirSubVentana()
      {
        nuevaVentana = window.open("ejemplo2_1.html", "sub",
        "height=300,width=400");
      }
     function cerrarSubVentana()
       {
         if (nuevaVentana) {
         nuevaVentana.close();
       }
     }
    </script>
  </head>
  <body>
    <h1>Ventana padre - principal</h1>
    <form action="">
    <p>
     <input type="button" value="Abrir sub ventana" id="abrir"
      onclick="abrirSubVentana()">
      <input type="button" value="Cerrar sub ventana" id="cerrar"
     onclick="cerrarSubVentana()">
    </p>
    <p>
      <label>Texto proveniente de la sub-ventana:</label>
      <input type="text" id="original">
    </p>
    </form>
    </body>
</html>
```

Ejemplo2_1.html

```
<html>
   <head>
     <meta http-equiv="content-type" content="text/html;charset=utf-8">
     <title>Sub-Documento</title>
     <script type="text/javascript">
       function copiarAlPadre()
        {
         opener.document.getElementById("original").value=
         document.getElementById('textocopiar').value;
        }
     </script>
   </head>
```

```
   <body>
     <h1>Sub-Ventana</h1>
     <form id="formulario">
       <p>
         <label for="textocopiar">Introduzca texto a copiar en la ventana
         principal:</label>
         <input type="text" id="textocopiar"/>
       </p>
       <p>
         <input type="button" value="Enviar texto a la ventana padre"
id="enviar"
         onclick="copiarAlPadre()" />
       </p>
    </form>
   </body>
</html>
```

Si no se abren las ventanas con el ejemplo anterior, a lo mejor tienes que desactivar el bloqueador de pop-ups y volver a probar.

4.-Objetos nativos en JavaScript.

Esto nos va a ser muy útil para aplicaciones que tendrán que usar diferentes tipos de conversiones de datos, trabajar intensivamente con cadenas y por supuesto con fechas y horas.

Aunque no hemos visto como crear objetos, sí que ya hemos dado unas pinceladas a lo que son los objetos, propiedades y métodos.

En esta sección vamos a echar una ojeada a objetos que son nativos en JavaScript, esto es, aquello que JavaScript nos da, listos para su utilización en nuestra aplicación.

Echaremos un vistazo a los objetos **`String`**, **`Math`**, **`Number`**, **`Boolean`** y **`Date`**.

4.1.- Objeto String.

Una cadena (`string`) consta de uno o más caracteres de texto, rodeados de comillas simples o dobles; da igual cuales usemos ya que se considerará una cadena de todas formas, pero en algunos casos resulta más cómodo el uso de unas u otras. Por ejemplo, si queremos meter el siguiente texto dentro de una cadena de JavaScript:

```
<input type="checkbox" name="coche" />Audi A6
```

Podremos emplear las comillas dobles o simples:

```
var cadena = '<input type="checkbox" name="coche" />Audi A6';
```

```
var cadena = "<input type='checkbox' name='coche' />Audi A6";
```

Si queremos emplear comillas dobles al principio y fin de la cadena, y que en el contenido aparezcan también comillas dobles, tendríamos que escaparlas con `\"`, por ejemplo:

```
var cadena = "<input type=\"checkbox\" name=\"coche\" />Audi A6";
```

Cuando estamos hablando de cadenas muy largas, podríamos concatenarlas con `+=`, por ejemplo:

```
var nuevoDocumento = "";
nuevoDocumento += "<!DOCTYPE html>";
nuevoDocumento += "<html>" ;
nuevoDocumento += "<head>";
nuevoDocumento += '<meta http-equiv="content-type"';
nuevoDocumento += ' content="text/html;charset=utf-8">';
```

Si queremos concatenar el contenido de una variable dentro de una cadena de texto emplearemos el símbolo `+` :

```
nombreEquipo = prompt("Introduce el nombre de tu equipo favorito:","");
var mensaje= "El " + nombreEquipo + " ha sido el campeón de la Copa del Rey!";
alert(mensaje);
```

Caracteres especiales o caracteres de escape.

La forma en la que se crean las cadenas en JavaScript, hace que cuando tengamos que emplear ciertos caracteres especiales en una cadena de texto, tengamos que escaparlos, empleando el símbolo \ seguido del carácter.

Vemos aquí un listado de los caracteres especiales o de escape en JavaScript:

Caracteres de escape y especiales en JavaScript	
Símbolos	**Explicación**
\”	Comillas dobles
\’	Comillas simples
\\	Barra inclinada
\b	Retroceso
\t	Tabulador
\n	Nueva línea
\r	Salto de línea
\f	Avance de pagina

4.1.1- Propiedades y métodos del objeto string.

Para crear un objeto `String` lo podremos hacer de la siguiente forma:

```
var miCadena = new String("texto de la cadena");
```

```
O también se podría hacer:
var miCadena = "texto de la cadena";
```

Es decir, cada vez que tengamos una cadena de texto, en realidad es un objeto `String` que tiene propiedades y métodos:

```
cadena.propiedad;
cadena.metodo( [parámetros] );
```

Propiedades del objeto string	
Propiedad	**Descripción**
`constructor`	Cea una instancia del objeto, el valor que se le dé dependerá de lo que se pase por parámetro.
`length`	Contiene el número de caracteres de la cadena.

Métodos del objeto string	
Métodos	**Descripción**
`anchor()`	Devuelve una cadena convertida en un ancla de HTML.
`big()`	Aumenta el tamaño de una cadena.
`blink()`	Crea el efecto de una cadena parpadeante.
`blod()`	Muestra una cadena en negrita.
`charAt()`	Devuelve el carácter especificado por la posición que se indica entre paréntesis.
`charCodeAt()`	Devuelve el Unicode del carácter especificado por la posición que se indica entre paréntesis.
`concat()`	Une una o más cadenas y devuelve el resultado de esa unión.
`fixed()`	Convierte una cadena con fuente monoespaciada.
`fontColor()`	Modifica el color de la fuente de una cadena.
`fontSize()`	Modifica e tamaño de una fuente de una cadena.
`fromCharCode()`	Convierte caracteres Unicode a caracteres.
`indexOf()`	Devuelve la posición de la primera ocurrencia del carácter buscado en la cadena.
`italics()`	Muestra la cadena en cursiva.
`lastIndexOf()`	Devuelve la última posición de ocurrencia del carácter buscado en la cadena.
`link()`	Muestra una cadena como un hipervínculo HTML con el enlace que le pasemos como parámetro.
`match()`	Busca una coincidencia entre una expresión regular y una cadena y devuelve las coincidencias o null si no ha encontrado nada.
`replace()`	Busca una subcadena en la cadena y la reemplaza por la nueva caena especificada.
`search()`	Busca una subcadena e la cadena y devuelve la posición donde la encontró.
`slice()`	Extrae una parte de la cadena y devuelve una nueva cadena.
`small()`	Disminuye el tamaño de una cadena.
`split()`	Divide una cadena en un array de subcadenas.
`substr()`	Extrae los caracteres de una cadena, comenzando en una determinada posición y con el número de caracteres indicado.
`substring`	Extrae los caracteres de una cadena entre dos índices indicados.
`toLowwerCase()`	Convierte una cadena en minúsculas.
`toUpCase()`	Convierte una cadena en mayúsculas.

Ejemplos de uso:

```
var cadena="El parapente es un deporte de riesgo medio";
document.write("La longitud de la cadena es: "+ cadena.length + "<br/>");
document.write(cadena.toLowerCase()+ "<br/>");
document.write(cadena.charAt(3)+ "<br/>");
document.write(cadena.indexOf('pente')+ "<br/>");
document.write(cadena.substring(3,16)+ "<br/>");
```

4.2.- Objeto Math.

Ya vimos anteriormente algunas funciones, que nos permitían convertir cadenas a diferentes formatos numéricos (`parseInt`, `parseFloat`). A parte de esas funciones, disponemos de un objeto `Math` en JavaScript, que nos permite realizar operaciones matemáticas. El objeto `Math` no es un constructor (no nos permitirá por lo tanto crear o instanciar nuevos objetos que sean de tipo `Math`), por lo que, para llamar a sus propiedades y métodos, lo haremos anteponiendo `Math` a la propiedad o el método. Por ejemplo:

```
var x = Math.PI; // Devuelve el número PI.
var y = Math.sqrt(16); // Devuelve la raíz cuadrada de 16.
```

Propiedades del objeto Math	
Propiedad	**Descripción**
`E`	Devuelve el número Euler (aprox. 2.718).
`LN2`	Devuelve el logaritmo neperiano de 2 (aprox. 0.693)
`LN10`	Devuelve el logaritmo neperiano de 10 (aprox 2.302)
`LOG2E`	Devuelve el logaritmo base 2 de E (aprox. 1.442)
`LOG10E`	Devuelve el logaritmo base 10 de E (aprox. 0.434)
`PI`	Devuelve el número PI de (aprox. 3.14159)
`SQRT2`	Devuelve la raíz cuadrada de 2 (aprox. 1.414)

Métodos del objeto Math	
Métodos	**Descripción**
`abs(x)`	Devuelve el valor absoluto de x.
`acos(x)`	Devuelve el arcocoseno de x en radianes .
`asin(x)`	Devuelve el arcoseno de x en radianes.
`atan(x)`	Devuelve el arcotangente de x en radianes, con un valor entre –PI/2 y PI/2.
`atan2(x,y)`	Devuelve el arcotangente del cociente de sus argumentos.
`ceil(x)`	Devuelve el número x redondeado al alta hacia el siguiente entero.
`cos(x)`	Devuelve el coseno de x (x está en radianes)
`floor(x)`	Devuelve el número x redondeado a la baja hacia el anterior entero.
`log(x)`	Devuelve el logaritmo neperiano (base E) de x.
`max(x, y, z,…,n)`	Devuelve el número más alto de los que se pasan por parámetro.
`min(x, y, z,…,n)`	Devuelve el número más bajo de los que se pasan por parámetro.

`pow(x, y)`	Devuelve le resultado de x elevado a y.
`random()`	Devuelve un número al azar entre 0 y 1.
`round(x)`	Redondea x al entero más próximo.
`sin(x)`	Devuelve el seno de x (x esta en radianes)
`sqrt(x)`	Devuelve la raíz cuadrada de x.
`tan(x)`	Devuelve la tangente de un ángulo.

Ejemplos de uso:

```
document.write(Math.cos(3) + "<br />");
document.write(Math.asin(0) + "<br />");
document.write(Math.max(0,150,30,20,38) + "<br />");
document.write(Math.pow(7,2) + "<br />");
document.write(Math.round(0.49) + "<br />");
```

4.3.- Objeto Number.

El objeto `Number` se usa muy raramente, ya que, para la mayor parte de los casos, JavaScript satisface las necesidades del día a día con los valores numéricos que almacenamos en variables. Pero el objeto `Number` contiene alguna información y capacidades muy interesantes para programadores más serios.

Lo primero, es que el objeto `Number` contiene propiedades que nos indican el rango de números soportados en el lenguaje. El número más alto es 1.79E + 308; el número más bajo es 2.22E-308.

Cualquier número mayor que el número más alto, será considerado como infinito positivo, y si es más pequeño que el número más bajo, será considerado infinito negativo.

Los números y sus valores están definidos internamente en JavaScript, como valores de doble precisión y de 64 bits.

El objeto `Number`, es un objeto envoltorio para valores numéricos primitivos.

La instrucción `Number()` da distintos resultados dependiendo del valor que se le pase:

Un número: el resultado es el mismo número:

```
var valor = Number(23) // valor = 23
```

Una cadena que representa un número: convierte el texto a número:

```
var valor = Number("42") // valor = 42
```

Una cadena con números y letras: el resultado es NaN, expresión que significa "Not a Number" (no es un número).

```
var valor = Number("13px") // valor = NaN
```

Una cadena con texto: el resultado es NaN

```
var valor = Number("un texto") // valor = NaN
```

Valor booleano true: el resultado es el número 1

```
var valor = Number(true) // valor = 1
```

Valor booleano false: el resultado es el número 0

```
var valor = Number(false) // valor = 0
```

Los objetos `Number` son creados con `new Number()`.

Propiedades del objeto Number	
Propiedad	**Descripción**
`constructor`	Devuelve la función que creo el objeto `Number`.
`MAX_VALUE`	Devuelve el número más alto disponible en JavaScript.
`MIN_VALUE`	Devuelve el número más bajo disponible en JavaScript.)
`NEGATIVE_INFINITE`	Representa el infinito negativo (se devuelve en caso de `overflow`)
`POSITIVE_INFINITE`	Representa el infinito POSITIVO (se devuelve en caso de `overflow`)
`prototype`	Permite añadir nuestras propias propiedades y métodos a un objeto.

Métodos del objeto Number	
Métodos	**Descripción**
`toExponential(x)`	Convierte un número a su notación exponencial.
`toFixed(x)`	Formatea un número con x dígitos decimales después del punto decimal.
`toPrecision(x)`	Formatea un número a la precisión x.
`toString()`	Convierte un objeto `Number` en una cadena. ➢ Si se pone 2 como parámetro se mostrará el número en binario. ➢ Si se pone 8 como parámetro se mostrará el número en octal. ➢ Si se pone 16 como parámetro se mostrará el número en hexadecimal.
`valueOf()`	Devuelve el valor primitivo de un objeto `Number`.

Ejemplos de uso:

```
var num = new Number(13.3714);
document.write(num.toPrecision(3)+"<br />");
document.write(num.toFixed(1)+"<br />");
document.write(num.toString(2)+"<br />");
document.write(num.toString(8)+"<br />");
document.write(num.toString(16)+"<br />");
document.write(Number.MIN_VALUE);
document.write(Number.MAX_VALUE);
```

4.4.- Objeto Boolean.

El objeto `Boolean` se utiliza para convertir un valor no Booleano, a un valor Booleano (`true` o `false`).

Propiedades del objeto Boolean	
Propiedad	**Descripción**
`constructor`	Devuelve la función que creo el objeto `Boolean`.
`prototype`	Permite añadir nuestras propias propiedades y métodos a un objeto.

Métodos del objeto Boolean	
Métodos	**Descripción**
`toString()`	Convierte un objeto `Boolean` en una cadena y devuelve el resultado.
`valueOf()`	Devuelve el valor primitivo de un objeto `Boolean`.

Ejemplos de uso:

```
var bool = new Boolean(1);
document.write(bool.toString());
document.write(bool.valueOf());
```

La clase `Boolean` es una clase nativa de JavaScript que nos permite crear valores booleanos.

Una de sus posibles utilidades es la de conseguir valores boleanos a partir de datos de cualquier otro tipo. No obstante, al igual que ocurría con la clase `Number`, es muy probable que no la llegues a utilizar nunca.

Dependiendo de lo que reciba el constructor de la clase `Boolean` el valor del objeto boleano que se crea será verdadero o falso, de la siguiente manera:

- **Se inicializa a `false`** cuando no pasas ningún valor al constructor, o si pasas una cadena vacía, el número 0 o la palabra `false` sin comillas.

- **Se inicializa a `true`** cuando recibe cualquier valor entrecomillado o cualquier número distinto de 0.

Se puede comprender el funcionamiento de este objeto fácilmente si examinamos unos ejemplos.

```
var b1 = new Boolean()
document.write(b1 + "<br>")
//muestra false
var b2 = new Boolean("")
document.write(b2 + "<br>")
//muestra false
var b25 = new Boolean(false)
document.write(b25 + "<br>")
//muestra false
```

```
var b3 = new Boolean(0)
document.write(b3 + "<br>")
//muestra false
var b35 = new Boolean("0")
document.write(b35 + "<br>")
//muestra true
var b4 = new Boolean(3)
document.write(b4 + "<br>")
//muestra true
var b5 = new Boolean("Hola")
document.write(b5 + "<br>")
//muestra true
```

4.5.- Objeto Date.

El objeto `Date` se utiliza para trabajar con fechas y horas. Los objetos `Date` se crean con `new Date ()`. Para obtener la representación de la fecha actual, sólo es necesario instanciar la clase sin parámetros:

Hay 4 formas de instanciar un objeto de tipo `Date`:

```
var d = new Date();
var d = new Date(milisegundos);
var d = new Date(cadena de Fecha);
var d = new Date(año, mes, día, horas, minutos, segundos, milisegundos);
// (el mes comienza en 0, Enero sería 0, Febrero 1, etc.)
```

Este objeto presenta una serie de métodos divididos en tres grupos:

- Métodos de lectura.
- Métodos de escritura.
- Métodos de ejecución.

Propiedades del objeto Date	
Propiedad	**Descripción**
`constructor`	Devuelve la función que creo el objeto `Date`.
`prototype`	Permite añadir nuestras propias propiedades y métodos a un objeto.

4.5.1.- Métodos del objeto date: lectura

Sin embargo, el objeto Date posee varios métodos que nos permiten ver, solo una parte de la fecha, por ejemplo, el día, el mes, o la hora.

Los métodos de lectura son distintos de los de escritura, todos ellos empiezan por la palabra `get`. Para aplicarlos escribimos la variable que almacena la fecha, seguida de un punto y el nombre del método: partimos de la siguiente variable:

```
fecha = new Date()
```

A partir de aquí vamos a ver los diferentes métodos de lectura; nosotros hemos puesto aquí la fecha actual, pero podríamos haber puesto cualquier otra fecha, si dentro del paréntesis hubiéramos escrito los parámetros anteriores. Una vez obtenemos la fecha mediante `fecha = new Date()` podemos aplicar los métodos de lectura:

Métodos del objeto Date: lectura.	
Métodos	**Descripción**
`getDate()`	Devuelve el día del mes (de 1-31).
`getDay()`	Devuelve el día de la semana (de 0-6).
`getFullYear()`	Devuelve el año en 4 dígitos.
`getHours()`	Devuelve la hora (de 0-23)
`getMilliseconds()`	Devuelve los milisegundos (de 0-999)
`getMinutes()`	Devuelve los minutos (0-59)
`getMonth()`	Devuelve el mes (0-11)
`getSeconds()`	Devuelve los segundos (0-59)
`getTime()`	Devuelve los milisegundos desde media noche del 1 de enero de 1970.
`getTimezoneOffset()`	Devuelve la diferencia de tiempo entre GMT y la hora local, en minutos.
`getUTCDate()`	Devuelve el día del mes en base a la hora UTC (1-31)
`getUTCDay()`	Devuelve el día de la semana en base a la hora UTC (0-6)
`getUTCFullYear()`	Devuelve el año en base a la hora UTC (4 dígitos)

4.5.2.- Métodos del objeto date: escritura.

Los métodos JavaScript para leer las fechas empezaban todos por la palabra get. Si cambiamos la palabra get por la palabra set, tenemos los métodos de escritura. El funcionamiento es idéntico a los métodos de lectura, es decir, debemos poner delante y separado por un punto la variable donde guardamos la fecha, sin embargo, dentro del paréntesis debemos poner el número o la variable que queremos modificar:

```
fecha = new Date();
nuevoDia = 1;
fecha.setDate(nuevoDia)
```

Métodos del objeto Date: escritura.	
Métodos	**Descripción**
`setDate()`	Cambia el día del mes recibido por parámetro.
`setMonth()`	Cambia el mes del año recibido por parámetro.
`setFullYear()`	Cambia el año recibido por parámetro. (escrito en 4 digito)
`setYear()`	Cambia el año recibido por parámetro.
`setHours()`	Cambia la hora recibida por parámetro.
`setMinutes()`	Cambia los minutos recibidos por parámetro.
`setSeconds()`	Cambia los segundos recibidos por parámetro.
`setMilisecond()`	Cambia los milisegundos recibidos por parámetro.
`setTimer()`	Cambia la fecha expresada en milisegundos a la recibida por parámetro.

4.5.3.- Métodos del objeto date: ejecución.

Sin embargo, muchas veces lo que queremos es controlar el tiempo que el usuario tarda en reaccionar ante un evento, o que una instrucción se ejecute pasado un cierto tiempo, o que la instrucción se repita cada cierto tiempo. Para ello JavaScript posee unos métodos bastante útiles, estos son los métodos `setTimeout()` y `setInterval()`. En realidad, estos métodos nada tienen que ver con el objeto `Date` visto en la página anterior; sino que dependen directamente del objeto `window`, por lo que no escribiremos nada delante.

Métodos del objeto Date: ejecución	
Métodos	**Descripción**
`setTimerOut()`	Hace que se ejecuten una serie de instrucciones contenidas dentro de una función después de transcurrido un cierto periodo de tiempo.
`setInterval()`	Hace que se repita una instrucción continuamente cada cierto intervalo de tiempo.
`clearTimeOut()`	Impide que se ejecute la instrucción indicada en `setTimeOut()` si aún no ha transcurrido el tiempo de espera.
`clearInterval()`	Detiene la ejecución de la repetición provocada por el método `setIntervel()`.

Ejemplos de uso:

```
var d = new Date();
document.write(d.getFullYear());
document.write(d.getMonth());
document.write(d.getUTCDay());
var d2 = new Date(5,28,2011,22,58,00);
d2.setMonth(0);
d.setFullYear(2020);
```

5.-Generacion de elementos HTML desde código JavaScript.

Uno de los principales objetivos de JavaScript en el desarrollo web en la parte del cliente es convertir un documento HTML estático en una aplicación web dinámica. Por ejemplo, es muy común que los scripts detecten el tipo y la versión del navegador que estamos utilizando y, en base a esto, escribir las etiquetas adecuadas para cada navegador. De igual modo, se suelen utilizar los valores de determinadas variables para ejecutar instrucciones que creen nuevas ventanas con contenido propio, en lugar de mostrarlo en la ventana actual, cada ventana de un navegador presenta un documento HTML y es representada por un objeto `Window` que contiene un subobjeto `Document`. El objeto `Document` contiene a su vez una serie de objetos que representan todo el contenido del documento HTML, como por ejemplo el texto, las imágenes, los enlaces, los formularios, etc.

Con JavaScript es posible manipular y acceder a los objetos que representan el contenido de una página. De este modo, en lugar de crear solamente documentos estáticos, es posible crear documentos dinámicos. Este proceso se puede realizar gracias al método `documen.write().`

Existen dos formas de utilizar este método para generar contenido dinámico:

- Podemos utilizarlo dentro de una secuencia de instrucciones JavaScript para mostrar un resultado en el documento de la ventana actual del navegador. Por ejemplo, es posible definir el título de una página web basándose en el nombre del SO que se utilice para abrir el documento:

```
<script type="text/javascript">
   var SO = navigator.platform;
   document.write("<h1>Documento abierto con: " + SO +
    "</h1">
</script>
```

- La segunda forma es muy similar a la anterior. El método `documen.write()` podemos utilizarlo para crear documentos de nuevas ventanas del navegador. Esto es una práctica muy común en las páginas web que utilizan ventanas emergentes. En el siguiente ejemplo de código podemos observar como creamos una nueva ventana sin ningún contenido, posteriormente accedemos a su respectivo documento y, finalmente, en la ventana nueva escribimos e título de la página0 web según el texto que ha ingresado el usuario:

```
<script type="text/javascript">
    var texto = prompt("Introduce un titulo para la
                        nueva ventana: ");
    varventanaNueva = window.open();
    ventanNueva.document.write("<h1>" + texto + "</h1>")
</script>
```

- La generación de código HTML a partir de código JavaScript no se limita solo a la creación de texto. Podemos crear y manipular todo tipo de objeto. El siguiente ejemplo muestra cómo generar un formulario para modificar la propiedad del color de fondo de la página.

```
<script type="text/javascript">
    document.write("<b>Selecciona un color para el fondo de
                página:</b><br>");
    document.write("<select name=\"color\">");
    document.write("<option value=\"red\"<Rojo</option>");
    document.write("<option value=\"blue\"<Azul</option>");
    document.write("<option value=\"yellow\"<Amarillo
                </option>");
    document.write("<option value=\"green\"<Verde</option>");

    document.write("<select>");
    document.write("<input type=\"button\" value=\"Modifica el color\"
      onclick=\"document.bgColor=document.cambiacolor.color.value\">");
    document.write("</form>");
</script>
```

6.- Gestión de ventanas.

Hemos visto cómo utilizar JavaScript para gestionar principalmente el contenido de las páginas web. Ahora vamos a ver cómo gestionar diferentes aspectos de las ventanas del navegador utilizando JavaScript.

Existen muchas operaciones comunes en las páginas web que visitamos a diario. Por ejemplo, es común que se abran ventanas nuevas al presionar un botón. Cada una de estas ventanas tiene tamaños, posiciones y estilos diferentes. Muchas de estas ventanas emergentes no contienen simplemente una página web estática. Su contenido suele ser dinámico y depende de las acciones realizadas en la ventana que la haya creado. Vamos a ver como abrir y cerrar nuevas ventanas, como controlar la apariencia de las mismas y como crear una comunicación interacción entre dos o más ventanas.

6.1- Abrir y cerrar nuevas ventanas.

Algo común que ocurre cuando navegamos por la Web es que se abra una ventana nueva cuando pulsamos un botón. Algunos sitios abren ventanas incluso sin que el usuario haga algo. Las ventanas se abren automáticamente cuando se carga una nueva página o simplemente cuando cerramos otra ventana.

Las ventanas secundarias las podemos abrir utilizando código HTML, mediante el atributo `target` de la etiqueta `href`:

```
<ahref="enlace.html" target="_blank">
```

La desventaja de abrir una nueva ventana utilizando solo código HTML, es que no tenemos ningún control sobre ella. La ventana se abrirá en base a la configuración predefinida del navegador del usuario. Por el contrario, con JavaScript tenemos un mayor control sobre las nuevas ventanas. Una nueva ventana vacía se crea con el método `open()`, el cual devuelve una referencia a dicha ventana:

```
nuevaVentana = window.open()
```

De este modo la variable `nuevaVentana` contendrá una referencia a la ventana que hemos creado. Esta variable la podemos utilizar posteriormente para cualquier manipulación que queramos hacer sobre la ventana mientras se ejecuta JavaScript.

Las ventanas poseen diversas propiedades y métodos tal y como hemos visto. En concreto el método `open()` cuenta con los siguientes tres parámetros:

- El primero es la URL de la página web que estará contenida en la nueva ventana.
- el segundo parámetro es el nombre asignado a la nueva ventana.
- el tercer parámetro es una colección de atributos que definen la apariencia de la ventana.

De este modo, el método `open()` lo podemos utilizar definiendo estos tres parámetros, tal y como podemos observar:

```
nuevaVentana = window.open(http://www.misitioWeb.com/ads, "Publicidad",
"height-100, width=100");
```

Utilizando los anteriores parámetros, abrimos una ventana que contiene la URL y el nombre especificados en los dos primeros parámetros y tiene una altura y anchura de 100 pixeles.

Vamos a ver un ejemplo completo de una página web en la que creamos un botón que abre una nueva ventana al hacer clic sobre él. En esta ventana estableceremos los atributos de altura y anchura, además de crear etiquetas HTML para generar el título de la ventana y un texto en el que especifiquemos las propiedades modificadas.

```
<html>
  <head> </head>
  <body>
   <center><h1> Ejemplo de Apariencia de una Ventana<h1>
    <br>
    <input type="Button" value="Abre una ventana"
          onClick="myWindow1=window.open("","Nueva Ventana",
          "width=300, height=200");
    myWindow.document.write("<html>");
    myWindow.document.write("<head>");
    myWindow.document.write("<title>Ventana de Prueba </title>");
    myWindow.document.write("</head>");
    myWindow.document.write("<body>");
```

```
    myWindow.document.writeln("Se han utilizado las propiedades:");
    myWindow.document.write("<li>height=200</li><li>width=300</li>");
    myWindow.document.write("</body>");
    myWindow.document.write("</html>");
    "/>
   </center>
  </body>
</html>
```

Cualquiera de las ventanas que abramos, podemos igualmente cerrarlas al invocar el método `close()`. Como hemos dicho anteriormente, en método `open()` devuelve una referencia a una nueva ventana. Para cerrar la ventana debemos invocar el método `close()` sobre dicha referencia.

Para probar su funcionamiento, podemos utilizar el código anterior y agregar justo antes de la creación de la etiqueta `</body>` la línea:

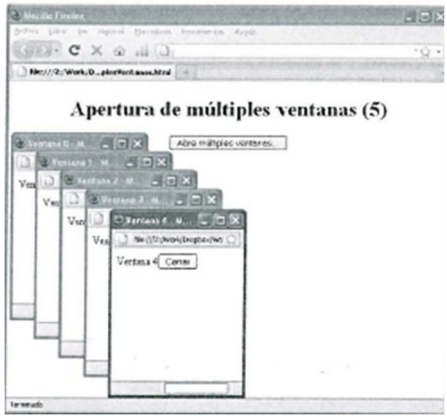
```
myWindow1.document.write("<input type-button
                        value=CerraronClick=window.close()>");
```

La apertura de múltiples ventanas a la vez es una característica por lo general bastante molesta para los usuarios. Sin embargo, en algunos casos puede ser útil saber cómo realizar esta operación. Para ello utilizamos un bucle `for` con el fin de ejecutar el método `window.open()` todas las veces deseadas, en el siguiente ejemplo abrimos cinco ventanas a la vez que presentan a su vez un botón para poder cerrarlas:

```
<html>
  <head> </head>
  <body>
   <center><h1>Apariencia de apertura de múltiples Ventanas<h1>
    <input     type="Button"     value="Abre     múltiples     ventanas"
       onClick="
       for (i=0; i<5; i++)
        {
          myWindow=window.open("","wihth=200,height=200");
          myWindow.document.write("<html>");
          myWindow.document.write("<head>");
          myWindow.document.write("<title>Ventana: ","+i" </title>);
          myWindow.document.write("</head>");
          myWindow.document.write("</head>");
          myWindow.document.write("<body>");
          myWindow.document.write("Ventana "+ i);
          myWindow.document.write("<input    type=button    value=Cerrar
               onClick=window.close()>");
          myWindow.document.write("</body>");
          myWindow.document.write("</html>");
         }
```

```
      "/>
    </center>
  </body>
</html>
```

En la siguiente figura podemos ver el resultado:



6.2- Apariencia de las ventanas.

Las ventanas cuentan con propiedades que podemos modificar con JavaScript. Estas propiedades permiten decidir el tamaño, la ubicación o los elementos que tendrá la ventana. Principales propiedades:

Propiedad	Descripción
`directories`	Corresponde a los botones del directorio estándar del navegador.
`height`	Corresponde a la altura de la ventana.
`menubar`	Corresponde a la barra del menú.
`resizable`	Corresponde a la opción de cambiar o no el tamaño de la ventana.
`scrollbar`	Corresponde a las barras de desplazamiento.
`status`	Corresponde a la barra de estado.
`toolbar`	Corresponde a la barra de herramientas.
`width`	Corresponde a la anchura de la ventana.

Las propiedades que definen la altura y la anchura se establecen determinando el tamaño en pixeles. Las demás propiedades sirven para decidir si mostrar o no un elemento de la ventana, así que los posibles valores que pueden tomar estas últimas propiedades son uno (1) o cero (0).

La ubicación de una nueva ventana la determina automáticamente el navegador. Sin embargo, podemos establecer la ubicación exacta de las ventanas

que abramos con JavaScript. Las propiedades `left` y `top` permiten definir esta ubicación y corresponden respectivamente a las coordenadas x e y en pixeles respecto a la esquina superior izquierda de la ventana. Los valores que definamos en estas propiedades debemos especificarlos en el tercer parámetro del método `open()`.

El siguiente ejemplo muestra una página web con un botón que permite la creación de una nueva ventana ubicada en la esquina superior izquierda de la pantalla. Esto ocurre ya que las propiedades `top` y `left` las ponemos iguales a cero. Además, establecemos una altura de 400 pixeles cada una.

```
<html>
  <head> </head>
  <body>
   <center><h1>Apariencia de las Ventanas<h1>
   <br>
    <input type="Button" value="Abre una ventana" onClick="
       myWindow1=window.open("","Nueva Ventana"," top=0, left=0,
         width=400, height=400");
       myWindow1.document.write("<html>);
       myWindow1.document.write("<head>);
       myWindow1.document.write("<title>Ubicar una aventana
         </title>");
       myWindow1.document.write("</head>);
       myWindow1.document.write("</body>);
       myWindow1.document.write("top=0 <br> left=0 <br> width=400,
           height=400");
       myWindow1.document.write("</body>);
       myWindow1.document.write("</html>);
    "/>
   </center>
  </body>
</html>
```

Debido a las diferentes resoluciones de pantalla los usuarios, no siempre se consigue el efecto deseado cuando definimos la ubicación de las nuevas ventanas. Para evitar este inconveniente, podemos especificar posiciones relativas utilizando el objeto `Screen` y averiguar previamente la resolución de la pantalla con las propiedades `screen.height` y `screen.width`. Una vez conozcamos los datos de la resolución, podemos utilizar porcentajes de estos mismos para definir las propiedades `left` y `top`.

6.3- Comunicación entre ventanas.

Hemos visto como desde una ventana se pueden abrir o cerrar nuevas ventanas. La primera se denomina ventana principal, mientras que las segundas se

denominan ventanas secundarias. La comunicación e interacción entre estas ventanas no es un proceso del todo bidireccional. Por ejemplo, una ventana principal puede abrir y cerrar la secundaria, pero por motivos de seguridad, no es posible realizar lo contrario.

Las nuevas ventanas se declaran como objetos y se les asigna un nombre. Gracias a este nombre, desde la ventana principal podemos acceder a ella y establecer una comunicación e interacción con sus elementos y propiedades, además de aplicar métodos de JavaScript, tal y como hemos visto anteriormente con el método `close()`. La ventana secundarias tiene el atributo `window.opener`, el cual hace referencia a la ventana principal. De este modo, desde las ventanas secundarias podemos acceder a los métodos y propiedades de la ventana principal.

En el siguiente ejemplo mostramos como acceder a la propiedad `location` del objeto `window` de una ventana secundaria. Esta propiedad contiene la URL del documento activo. La ventana principal cuenta con un formulario que presenta un campo de texto en el que podemos escribir una URL, además de un botón que al presionarlo cambia la propiedad `location` de la ventana secundaria. Esta última ventana mostrará la URL que haya escrito el usuario.

```
<html>
  <head> </head>
  <body>
   <script>
      Var     ventanaSecundadria=window.open(“”,”VentanaSec”,”width=500,
                height=500”);
   </script>
   <center><h1>Comunicación entre Ventanas<h1>
     <br>
     <form name=formulario>
        <input type=text name=url size=50 value=http://www.>
        <input    type=”Button”    value=”Mostrar    URL    en    ventana
             secundaria”      onClick=”ventanaSecundaria.location      =
            document.formulario.url.value;”>
     </form>
   </center>
  </body>
<html>
       myWindow1.document.write(“<html>);
```

ANEXO I – EL OBJETO WINDOW.

Se trata del objeto **más alto en la jerarquía del navegador** (`navigator` es un objeto independiente de todos en la jerarquía), pues todos los componentes de una página web están situados dentro de una ventana. El objeto `window` hace referencia a la ventana actual. Veamos a continuación sus propiedades y sus métodos.

Propiedades:

- ✓ **closed:** válida a partir de Netscape 3 en adelante y MSIE 4 en adelante. Es un booleano que nos dice si la ventana está cerrada (`closed = true`) o no (`closed = false`).
- ✓ **`defaultStatus`:** cadena que contiene el texto por defecto que aparece en la barra de estado (status bar) del navegador.
- ✓ **`frames`:** es un array: cada elemento de este array (`frames[0], frames[1], ...`) es uno de los frames que contiene la ventana. Su orden se asigna según se definen en el documento HTML.
- ✓ **`history`:** se trata de un array que representa las URLS visitadas por la ventana (están almacenadas en su historial).
- ✓ **`length`:** variable que nos indica cuántos `frames` tiene la ventana actual.
- ✓ **`location`:** cadena con la URL de la barra de dirección.
- ✓ **`name`:** contiene el nombre de la ventana, o del `frame` actual.
- ✓ **`opener`**: es una referencia al objeto `window` que lo abrió, si la ventana fue abierta usando el método `open()` que veremos cuando estudiemos los métodos.
- ✓ **`parent`:** referencia al objeto `window` que contiene el `frameset`.
- ✓ **`self`:** es un nombre alternativo del `window` actual.
- ✓ **`status`**. `string` con el mensaje que tiene la barra de estado.
- ✓ **`top`:** nombre alternativo de la ventana del nivel superior.
- ✓ **`window`:** igual que `self`: nombre alternativo del objeto `window` actual.

Métodos:

- ✓ **`alert(mensaje)`**: muestra el mensaje 'mensaje' en un cuadro de diálogo
- ✓ **`blur()`:** elimina el foco del objeto `window` actual. A partir de NS 3, IE 4.
- ✓ **`clearInterval(id)`:** elimina el intervalo referenciado por 'id' (ver el método `setInterval()`, también del objeto `window`). A partir de NS 4, IE 4.
- ✓ **`clearTimeout(nombre)`:** cancela el intervalo referenciado por 'nombre' (ver el método `setTimeout()`, también del objeto `window`).
- ✓ **`close()`:** cierra el objeto `window` actual.
- ✓ **`confirm(mensaje)`:** muestra un cuadro de diálogo con el mensaje 'mensaje' y dos botones, uno de aceptar y otro de cancelar. Devuelve true si se pulsa aceptar y devuelve false si se pulsa cancelar.
- ✓ **`focus()`:** captura el foco del ratón sobre el objeto `window` actual. A partir de NS 3, IE 4.
- ✓ **`moveBy(x,y)`:** mueve el objeto `window` actual el número de pixels especificados por (x,y). A partir de NS 4.

- ✓ **`moveTo(x,y):`** mueve el objeto `window` actual a las coordenadas (x,y). A partir de NS 4.
- ✓ **`open(URL,nombre,caracteristicas`**): abre la URL que le pasemos como primer parámetro en una ventana de nombre 'nombre'. Si esta ventana no existe, abrirá una ventana nueva en la que mostrará el contenido con las características especificadas. Las características que podemos elegir para la ventana que queramos abrir son las siguientes:
  - → `toolbar = [yes|no|1|0].` Nos dice si la ventana tendrá barra de herramientas (yes,1) o no la tendrá (no,0).
  - → `location = [yes|no|1|0].` Nos dice si la ventana tendrá campo de localización o no.
  - → `directories = [yes|no|1|0].` Nos dice si la nueva ventana tendrá botones de dirección o no.
  - → `status = [yes|no|1|0].` Nos dice si la nueva ventana tendrá barra de estado o no.
  - → `menubar = [yes|no|1|0].` Nos dice si la nueva ventana tendrá barra de menús o no.
  - → `scrollbars = [yes|no|1|0].` Nos dice si la nueva ventana tendrá barras de desplazamiento o no.
  - → `resizable = [yes|no|1|0].` Nos dice si la nueva ventana podrá ser cambiada de tamaño (con el ratón) o no.
  - → `width = px`. Nos dice el ancho de la ventana en pixels.
  - → `height = px`. Nos dice el alto de la ventana en pixels.
  - → `outerWidth = px`. Nos dice el ancho ***total*** de la ventana en pixels. A partir de NS 4.
  - → outerHeight = px. Nos dice el alto ***total*** de la ventana el pixels. A partir de NS 4
  - → `left = px.` Nos dice la distancia en pixels desde el lado izquierdo de la pantalla a la que se debe colocar la ventana.
  - → `top = px.` Nos dice la distancia en pixels desde el lado superior de la pantalla a la que se debe colocar la ventana.
- ✓ **`prompt(mensaje,respuesta_por_defecto)`**: muestra un cuadro de diálogo que contiene una caja de texto en la cual podremos escribir una respuesta a lo que nos pregunte en '*mensaje*'. El parámetro '*respuesta_por_defecto*' es opcional, y mostrará la respuesta por defecto indicada al abrirse el cuadro de diálogo. El método retorna una cadena de caracteres con la respuesta introducida.
- ✓ **`scroll(x,y)`**: desplaza el objeto `window` actual a las coordenadas especificadas por (x,y). A partir de NS3, IE4.
- ✓ **`scrollBy(x,y)`**: desplaza el objeto `window` actual el número de pixels especificado por (x,y). A partir de NS4.
- ✓ **`scrollTo(x,y)`**: desplaza el objeto `window` actual a las coordenadas especificadas por (x,y). A partir de NS4.
- ✓ **`setInterval(expresion,tiempo)`**: evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un

valor que puede ser usado como identificativo por `clearInterval()`. A partir de NS4, IE4.

- ✓ **`setTimeout(expresion,tiempo)`**: evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por `clearTimeout()`. A partir de NS4, IE4.

No están todas las propiedades y métodos como innerHeight, innerWidth, outerHeight, outerWidth, pageXOffset, pageYOffset, personalbar, scrollbars, back(), find(["cadena"],[caso,bkwd]), forward(), home(), print(), stop()... todas ellas disponibles a partir de NS 4. Realizaremos ejercicios donde se irán viendo más propiedades y métodos.

Ejemplo:

```
<HTML>
  <HEAD>
    <title>Ejemplo de JavaScript</title>
    <script LANGUAGE="JavaScript">
    <!--
    function moverVentana()
    {
      mi_ventana.moveBy(5,5);
      i++;
      if (i<20)
        setTimeout('moverVentana()',100);
      else
      mi_ventana.close();
    }
    //-->
    </script>
  </HEAD>
  <BODY>
    <script LANGUAGE="JavaScript">
    <!--
      var opciones="left=100,top=100,width=250,height=150", i= 0;
      mi_ventana = window.open("","",opciones);
      mi_ventana.document.write("Una prueba de abrir ventanas");
      mi_ventana.moveTo(400,100);
      moverVentana();
    //-->
    </script>
   </BODY>
</HTML>
```

ANEXO II – EL OBJETO LOCATION.

Este objeto contiene la URL actual, así como algunos datos de interés respecto a esta URL. Su finalidad principal es, por una parte, modificar el objeto `location` para cambiar a una nueva URL, y extraer los componentes de dicha URL de forma separada para poder trabajar con ellos de forma individual si es el caso. Recordemos que la sintaxis de una URL era:

```
protocolo://maquina_host[:puerto]/camino_al_recurso
```

Propiedades:

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

Métodos:

- ✓ **`reload()`**: vuelve a cargar la URL especificada en la propiedad href del objeto location.
- ✓ **`replace(cadenaURL)`**: reemplaza el historial actual mientras carga la URL especificada en cadenaURL.

Ejemplo:

```
<HTML>
  <HEAD>
    <title>Ejemplo de JavaScript</title>
  </HEAD>
  <BODY>
    <script LANGUAGE="JavaScript">
    <!--
      document.write("Location   <b>href</b>:   "   +   location.href   +
                                  "<br>");
      document.write("Location   <b>host</b>:   "   +   location.host   +
                        "<br>");
      document.write("Location <b>hostname</b>: " + location.hostname +
                       "<br>");
      document.write("Location <b>pathname</b>: " + location.pathname +
                       "<br>");
      document.write("Location   <b>port</b>:   "   +   location.port   +
           "<br>");
      document.write("Location <b>protocol</b>: " + location.protocol +
                        "<br>");
    //-->
    </script>
```

```
  </BODY>
</HTML>
```

ANEXO III – EL OBJETO NAVIGATOR.

Este objeto simplemente nos da información relativa al navegador que esté utilizando el usuario.

Propiedades:

- ✓ **`appCodeName`**: cadena que contiene el nombre del código del cliente.
- ✓ **`appName`**: cadena que contiene el nombre del cliente.
- ✓ **`appVersion`**: cadena que contiene información sobre la versión del cliente.
- ✓ **`language`**: cadena de dos caracteres que contiene información sobre el idioma de la versión del cliente.
- ✓ **`mimeTypes`**: Array que contiene todos los tipos MIME soportados por el cliente. A partir de NS 3.
- ✓ **`platform`**: cadena con la plataforma sobre la que se está ejecutando el programa cliente.
- ✓ **`plugins`**: Array que contiene todos los plug-ins soportados por el cliente. A partir de NS 3.
- ✓ **`userAgent`**: cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades `appCodeName` y `appVersion`.

Métodos:

- ✓ **`javaEnabled()`** :devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.

Ejemplo:

```
<HTML>
  <HEAD>
    <title>Ejemplo de JavaScript</title>
  </HEAD>
  <BODY>
    <script LANGUAGE="JavaScript">
    <!--
      document.write("Navigator       <b>appCodeName</b>:       "       +
                     navigator.appCodeName + "<br>");
      document.write("Navigator <b>appName</b>: " + navigator.appName +
                     "<br>");
      document.write("Navigator       <b>appVersion</b>:       "       +
                      navigator.appVersion + "<br>");
      document.write("Navigator <b>language</b>: " + navigator.language
                     + "<br>");
      document.write("Navigator <b>platform</b>: " + navigator.platform
+  "<br>");
      document.write("Navigator       <b>userAgent</b>:       "       +
                      navigator.userAgent + "<br>");
    //-->
    </script>
  </BODY>
</HTML>
```

ANEXO IV – EL OBJETO STRING.

El objeto `String` se utiliza para manipular una cadena almacenada de texto.

Los objetos `String` se crean con `new String()`

Sintaxis:

```
var txt = new String("cadena");
```

o simplemente:

```
var txt = "cadena";
```

Propiedades:

- ✓ **`constructor`**: devuelve la función que ha creado el prototipo del objeto String
- ✓ **`length`**: devuelve la longitud de una cadena
- ✓ **`prototype`**: te permite añadir propiedades y métodos a un objeto

Métodos del objeto String:

- ✓ **`charAt()`**: devuelve el carácter en el índice especificado.
- ✓ **`charCodeAt()`**: devuelve el carácter Unicode del índice especificado.
- ✓ **`concat()`**: une dos o más cadenas y devuelve una copia de las cadenas unidas.
- ✓ **`fromCharCode()`**: convierte valores Unicode a caracteres.
- ✓ **`indexOf()`**: devuelve la posición de la primera aparición de un valor especificado en una cadena.
- ✓ **`lastIndexOf()`** Devuelve la posición de la última aparición de un valor especificado en una cadena.
- ✓ **`match()`**: busca una coincidencia entre una expresión regular y una cadena, y devuelve las coincidencias.
- ✓ **`replace()`**: busca una coincidencia entre una subcadena (o expresión regular) y una cadena, y sustituye a la subcadena encontrada con una nueva subcadena.
- ✓ **`search()`**: busca una coincidencia entre una expresión regular y una cadena, y devuelve la posición de la coincidencia.
- ✓ **`slice()`**: extrae una parte de una cadena y devuelve una nueva cadena.
- ✓ **`split()`**: divide una cadena dentro de un array de subcadenas.
- ✓ **`substr()`**: extrae los caracteres de una cadena, empezando en la posición de inicio especificado, y el número especificado de caracteres
- ✓ **`substring()`**: extrae los caracteres de una cadena, entre dos índices especificados.
- ✓ **`toLowerCase()`**: convierte una cadena a minúsculas.
- ✓ **`toUpperCase()`**: convierte una cadena a mayúsculas.
- ✓ **`valueOf()`**: devuelve el valor primitive de un objeto String.

Métodos envolventes de String HTML:

Los métodos envolventes HTML devuelven la cadena envuelta en una etiqueta HTML apropiada

- ✓ `anchor()`: crea un anchor (ancla).
- ✓ `big()`: visualiza una cadena usando una fuente grande.
- ✓ `blink()`: visualiza una cadena parpadeante.
- ✓ `bold()`: visualiza una cadena en negrita.
- ✓ `fixed()`: visualiza una cadena utilizando una fuente de paso fijo.
- ✓ `fontcolor()`: visualiza una cadena usando el color especificado.
- ✓ `fontsize()`: visualiza una cadena usando el tamaño especificado.
- ✓ `italics()`:visualiza una cadena en cursiva.
- ✓ `link()`: visualiza una cadena como un vínculo.
- ✓ `small()`: visualiza una cadena en letra pequeña.
- ✓ `strike()`: visualiza una cadena con un tachado.
- ✓ `sub()`: visualiza una cadena como texto subíndice.
- ✓ `sup()`:visualiza una cadena como texto superíndice.