Modelo de objetos predefinidos en JavaScript

Tabla de contenido

1.- Objetos de más alto nivel en JavaScript.

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 has visto en la unidad anterior, 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.

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] )
```

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 encontrarás frecuentemente en tus aplicaciones de JavaScript: **`window`**, **`location`**, **`navigator`** y **`document`**. El objetivo, no es solamente ver 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 encontrarás en unidades posteriores.

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

Te mostramos aquí el gráfico del modelo de objetos de alto nivel, para todos los navegadores que permitan usar JavaScript.

Es muy importante que tengas este gráfico en mente porque va a ser la guía a lo largo de toda esta unidad.

1.1.- 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.

1.1.1.- 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, si que podrá crear o abrir nuevas subventanas.

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=600,width=800");
```

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.

Véase el siguiente ejemplo que permite abrir y cerrar una subventana:

```
<!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">
/* 2º DAW Colegio Montessori Zaragoza Desarrollo WEB en entorno cliente
Ejecicio objeto Window */

function inicializar(){
document.getElementById("crear-ventana").onclick=crearNueva;
document.getElementById("cerrar-ventana").onclick=cerrarNueva;
}

var  nuevaVentana;  // importante declararla aquí
function crearNueva(){
nuevaVentana = window.open("http://www.fundacionmontessori.com","","height=600,width=800");
}

function cerrarNueva(){
if (nuevaVentana){
nuevaVentana.close();
nuevaVentana  =  null;
}
}
</script>
</head>
<body onLoad="inicializar()">
<h1>Apertura y cierre de 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>
```


El mismo ejercicio realizado de otra manera

```
<!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">
/*
2º DAW Colegio Montessori Zaragoza Desarrollo WEB en entorno cliente
Ejecicio objeto Window
*/
var  nuevaVentana; // importante declararla aquí
function crearNueva(){
nuevaVentana = window.open("http://www.fundacionmontessori.com","","height=600,width=800");
}

function cerrarNueva(){
if (nuevaVentana){
nuevaVentana.close();
nuevaVentana = null;
}
}
</script>
</head>
<body>
<h1>Apertura y cierre de ventanas</h1>
<form>
<p> <input type="button" id="crear-ventana"  value="Crear Nueva Ventana" onclick=crearNueva()>
<input type="button" id="cerrar-ventana"  value="Cerrar Nueva Ventana" onclick=cerrarNueva()> </p>
</form>
</body>
</html>
```

1.1.2.- Propiedades y métodos.

El objeto Window representa una ventana abierta en un 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.
length	**Devuelve el número de frames (incluyendo iframes) que hay en dentro de una ventana.**
location	**Devuelve la Localización del objeto ventana (URL del fichero).**
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.
parent	Devuelve la ventana padre de la ventana actual.
self	Devuelve la ventana actual.
status	Ajusta el texto de la barra de estado de una ventana.

Métodos del objeto Window

Método	Descripción
alert()	Muestra una ventana emergente de alerta y un botón de aceptar.
blur()	Elimina el foco de la ventana actual.
clearInterval()	Resetea el cronómetro ajustado con `setInterval()`.
setInterval()	Llama a una función o evalúa una expresión en un intervalo especificado (en milisegundos).
close()	Cierra la ventana actual.
confirm()	Muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.

`focus()`	Coloca el foco en la ventana actual.
`open()`	Abre una nueva ventana de navegación.
`prompt()`	Muestra una ventana de diálogo para introducir datos.

En el siguiente anexo puedes ampliar la información sobre el objeto **Window** y todas sus propiedades y métodos.

Ejercicio

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>CRONOMETRO</title>
    <SCRIPT LANGUAGE="JavaScript">
        var intervalo;
        function iniciar(){
            intervalo = window.setInterval(iniciarContador, 1000); // Necesita
minimo dos parametros, la funcion a dispara y el intervalo en el que va a ser
llamamada en ms
        }
        function iniciarContador(){
            document.form1.reloj.value = parseInt(document.form1.reloj.value) +
1;
        }
        function pararContador(){
            window.clearInterval(intervalo);
        }
    </SCRIPT>
</head>
<body>
<CENTER><H1>C R O N O M E T R O</H1></CENTER>
<FORM name="form1">
    <center><INPUT type="text" id="reloj" value=0><center><br>
    <input type="button" id="boton1" value="INICIO" onClick="iniciar()">
    <input type="button" id="boton2" VALUE="PARAR" onClick="pararContador()">
</FORM>
</body>
</html>
```

1.2.- 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 puntos), dentro de la URL.

`search`	Cadena que contiene la información pasada en una llamada a un script, dentro de la URL.

Métodos del objeto Location

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

1.3.- 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 en 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 plataforma 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

Método	Descripción
javaEnabled()	Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.

1.4.- 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).

Colecciones del objeto Document

Colección	Descripción
`anchors[]`	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 todas las imágenes del documento.
`links[]`	Es un array que contiene todos los enlaces del documento.

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 cargó el documento.
`lastModified`	Devuelve la fecha y hora de la última modificación del documento
`readyState`	Devuelve el estado de carga del documento actual
`referrer`	Cadena que contiene la URL del documento desde el cuál llegamos al documento actual.
`title`	Devuelve o ajusta el título del documento.
`URL`	Devuelve la URL completa del documento.

Método	Descripción
`close()`	Cierra el flujo abierto previamente con document.open().
`getElementById()`	Para acceder a un elemento identificado por el id escrito entre paréntesis.
`getElementsByName()`	Para acceder a los elementos identificados por el atributo name escrito entre paréntesis.
`getElementsByTagName()`	Para acceder a los elementos identificados por el tag o la etiqueta escrita entre paréntesis.
`open()`	Abre el flujo de escritura para poder utilizar document.write() o document.writeln en el documento.
`write()`	Para poder escribir expresiones HTML o código de JavaScript dentro de un documento.
`writeln()`	Lo mismo que write() pero añade un salto de línea al final de cada instrucción.

1.5. Modelo básico de eventos

1.5.1. Tipos de eventos

En este modelo, cada elemento o etiqueta XHTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos XHTML diferentes y un mismo elemento XHTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo `on`, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina `onclick` y el evento asociado a la acción de mover el ratón se denomina `onmousemove`.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

Evento	Descripción	Elementos para los que está definido
`onblur`	Deseleccionar el elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Deseleccionar un elemento que se ha modificado	`<input>`, `<select>`, `<textarea>`
`onclick`	Pinchar y soltar el ratón	Todos los elementos
`ondblclick`	Pinchar dos veces seguidas con el ratón	Todos los elementos
`onfocus`	Seleccionar un elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onkeydown`	Pulsar una tecla (sin soltar)	Elementos de formulario y `<body>`

Evento	Descripción	Elementos para los que está definido
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Los eventos más utilizados en las aplicaciones web tradicionales son `onload` para esperar a que se cargue la página por completo, los eventos `onclick`, `onmouseover`, `onmouseout` para controlar el ratón y `onsubmit` para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (`onclick`, `onkeydown`, `onkeypress`, `onreset`, `onsubmit`) permiten evitar la *"acción por defecto"* de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo `<input type="submit">` se desencadenan los eventos `onmousedown`, `onclick`, `onmouseup` y `onsubmit` de forma consecutiva.

1.5.2. Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede *responder* ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan *"manejador de eventos"* y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos XHTML.
- Manejadores como funciones JavaScript externas.
- Manejadores *"semánticos"*.

1.5.2.1. Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez *menos profesional* de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo

del propio elemento XHTML. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por
pinchar');" />
```

En este método, se definen atributos XHTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es `onclick`. Así, el elemento XHTML para el que se quiere definir este evento, debe incluir un atributo llamado `onclick`.

El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. En este caso, el código JavaScript es muy sencillo (`alert('Gracias por pinchar');`), ya que solamente se trata de mostrar un mensaje.

En este otro ejemplo, cuando el usuario pincha sobre el elemento `<div>` se muestra un mensaje y cuando el usuario pasa el ratón por encima del elemento, se muestra otro mensaje:

```
<div onclick="alert('Has pinchado con el ratón');" onmouseover="alert('Acabas
de pasar el ratón por encima');">
  Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima
</div>
```

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas:

```
<body onload="alert('La página se ha cargado completamente');">
   ...
</body>
```

El mensaje anterior se muestra después de que la página se haya cargado completamente, es decir, después de que se haya descargado su código HTML, sus imágenes y cualquier otro objeto incluido en la página.

El evento `onload` es uno de los más utilizados ya que, como se vio en el capítulo de DOM, las funciones que permiten acceder y manipular los nodos del árbol DOM solamente están disponibles cuando la página se ha cargado completamente.

1.5.2.2. Manejadores de eventos y variable this

JavaScript define una variable especial llamada `this` que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable `this` para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del `<div>`, el color del borde se muestra de color negro. Cuando el ratón *sale* del `<div>`, se vuelve a mostrar el borde con el color gris claro original.

Elemento `<div>` original:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid
silver">
  Sección de contenidos...
</div>
```

Si no se utiliza la variable `this`, el código necesario para modificar el color de los bordes, sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid
silver"
onmouseover="document.getElementById('contenidos').style.borderColor='black';"
onmouseout="document.getElementById('contenidos').style.borderColor='silver';"
>
  Sección de contenidos...
</div>
```

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable `this`, que hace referencia al elemento XHTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid
silver" onmouseover="this.style.borderColor='black';"
onmouseout="this.style.borderColor='silver';">
  Sección de contenidos...
</div>
```

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente aunque se modifique el valor del atributo `id` del `<div>`.

1.5.2.3. Manejadores de eventos como funciones externas

La definición de los manejadores de eventos en los atributos XHTML es el método más sencillo pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:


```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por
pinchar');" />
```

Utilizando funciones externas se puede transformar en:

```
function muestraMensaje() {
  alert('Gracias por pinchar');
}

<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable `this` y por tanto, es necesario pasar esta variable como parámetro a la función:

```
function resalta(elemento) {
  switch(elemento.style.borderColor) {
    case 'silver':
    case 'silver silver silver silver':
    case '#c0c0c0':
      elemento.style.borderColor = 'black';
      break;
    case 'black':
    case 'black black black black':
    case '#000000':
      elemento.style.borderColor = 'silver';
      break;
  }
}

<div style="width:150px; height:60px; border:thin solid silver"
onmouseover="resalta(this)" onmouseout="resalta(this)">
  Sección de contenidos...
</div>
```

En el ejemplo anterior, la función externa es llamada con el parámetro `this`, que dentro de la función se denomina `elemento`. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad `borderColor`.

Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor `black`, Internet Explorer lo almacena como `black black black black` y Opera almacena su representación hexadecimal `#000000`.

1.5.2.4. Manejadores de eventos semánticos

Los métodos que se han visto para añadir manejadores de eventos (como atributos XHTML y como funciones externas) tienen un grave inconveniente: *"ensucian"* el código XHTML de la página.

Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (XHTML) y su aspecto o presentación (CSS). Siempre que sea posible, también se recomienda separar los contenidos (XHTML) y su comportamiento o programación (JavaScript).

Mezclar el código JavaScript con los elementos XHTML solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

```
<input id="pinchable" type="button" value="Pinchame y verás"
onclick="alert('Gracias por pinchar');" />
```

Se puede transformar en:

```
// Función externa
function muestraMensaje() {
  alert('Gracias por pinchar');
}

// Asignar la función externa al elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Elemento XHTML
<input id="pinchable" type="button" value="Pinchame y verás" />
```

La técnica de los manejadores semánticos consiste en:

1. Asignar un identificador único al elemento XHTML mediante el atributo `id`.
2. Crear una función de JavaScript encargada de manejar el evento.
3. Asignar la función externa al evento correspondiente en el elemento deseado.

El último paso es la clave de esta técnica. En primer lugar, se obtiene el elemento al que se desea asociar la función externa:

```
document.getElementById("pinchable");
```

A continuación, se utiliza una propiedad del elemento con el mismo nombre que el evento que se quiere manejar. En este caso, la propiedad es `onclick`:

```
document.getElementById("pinchable").onclick = ...
```

Por último, se asigna la función externa mediante su nombre sin paréntesis. Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función:

```
document.getElementById("pinchable").onclick = muestraMensaje;
```


Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad `onclick` de elemento.

```
// Asignar una función externa a un evento de un elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Ejecutar una función y guardar su resultado en una propiedad de un elemento
document.getElementById("pinchable").onclick = muestraMensaje();
```

La gran ventaja de este método es que el código XHTML resultante es muy *"limpio"*, ya que no se mezcla con el código JavaScript. Además, dentro de las funciones externas asignadas sí que se puede utilizar la variable `this` para referirse al elemento que provoca el evento.

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento `onload`:

```
window.onload = function() {
  document.getElementById("pinchable").onclick = muestraMensaje;
}
```

La técnica anterior utiliza el concepto de *funciones anónimas*, que no se va a estudiar, pero que permite crear un código compacto y muy sencillo. Para asegurarse que un código JavaScript va a

ejecutarse después de que la página se haya cargado completamente, sólo es necesario incluir esas instrucciones entre los símbolos { y }:

```
window.onload = function() {
  ...
}
```

En el siguiente ejemplo, se añaden eventos a los elementos de tipo `input=text` de un formulario complejo:

```
function resalta() {
  // Código JavaScript
}

window.onload = function() {
  var formulario = document.getElementById("formulario");
  var camposInput = formulario.getElementsByTagName("input");

  for(var i=0; i<camposInput.length; i++) {
    if(camposInput[i].type == "text") {
      camposInput[i].onclick = resalta;
    }
  }
}
```

2.- Marcos.

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

Para cada etiqueta **`<frame>`** en un documento HTML, se creará un objeto **`frame`**.

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

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 un `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 `iframe`.

Eventos del objeto Frame/Iframe	
Evento	**Descripción**
`onload`	Script que se ejecutará inmediatamente después a que se cargue el `frame`/`iframe`.

2.1.- 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 te daremos algunas nociones para trabajar con múltiples ventanas y marcos.

Marcos: Padres e Hijos.

En el gráfico de jerarquías de objetos, viste 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).

Véase la siguiente figura de 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 sítienen objeto **document**. El objeto **document** de un marco es independiente del objeto **document** de otro marco, y en realidad cada uno de los marcos, será un objeto **window** independiente.

2.2.- Comunicación entre marcos.

Referencias Padre-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
```


2.3.- 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.

Si queremos comunicar dos marcos que están en una misma ventana lo haremos:

A través de su padre (parent).

Directamente con el nombre del marco.

A través del objeto navigator.

A través de parent podremos conectar dos marcos hijos para poder acceder a las diferentes propiedades u objetos de cada uno de ellos.

3.- Objetos nativos en Javascript.

Esto le va a ser muy útil para realizar su aplicación ya que tendrá que realizar 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`.

3.1 – Expresiones Regulares

Una expresión regular es un patrón que se emplea para compararse con un grupo de caracteres •

Una expresión regular, llamada también patrón, es una expresión que describe un conjunto de cadenas sin enumerar normalmente sus elementos.

Un ejemplo ilustrativo sería el grupo formado por las cadenas Handel, Händel y Haendel se describe mediante el patrón "H(a|ä|ae)ndel" o por las cadenas Camion, Camión mediante la expresión regular “Cami(o|ó)n”.

3.1.1 – Introducción

Una expresión regular es un patrón que puede estar formado por un conjunto de caracteres (letras, números o signos) y por un conjunto de metacaracteres que representan otros caracteres o que indican la forma de combinar los caracteres.

Las expresiones regulares se pueden emplear en:
– Comandos de sistemas operativos, como sed y grep en Linux
– Editores de texto avanzados
– Lenguajes de programación, como JavaScript, PHP y Perl, o a través de librerías como Java o .NET.
– Existen dos estilos de expresiones regulares.

La sintaxis de una expresión regular debe empezar y terminar por “/”,por ejemplo: / ……/.

- `/a+/` : La cadena debe contener el patrón (letra "a") una o más veces.
- `/a*/` : El patrón (letra a) debe aparecer cero o más veces.
- `/a?/` : El patón debe aparecer cero o una vez.
- `/a{3}/` : El patón debe aparecer exactamente tres veces (en lugar del 3 puede ser cualquier otro número).
- `/a{3,8}/` : El patón debe aparecer entre 3 y 8 veces (ambos inclusive).

Las expresiones regulares usan metacaracteres que reciben este nombre porque no se representan a ellos mismos, sino que son interpretados de una manera concreta.

El objeto String (que se verá a continuación) tiene unos métodos que admiten Expresiones Regulares:
– search(): busca en una cadena la ER, devuelve la posición
– match(): busca en una cadena la ER, devuelve un array con los valores emparejados

– replace(): sustituye unos caracteres por otros caracteres
– split(): divide una cadena en un array de cadenas

3.1.2 - Los metacaracteres más usados son: * ? + [ ] ( ) { } ^ $ | ...

– **^**: Sirve para indicar que el patrón que lo acompaña esta al principio de la cadena.
– **$**: Indica que el patrón esta al final de una cadena.
– **.**: Representa cualquier carácter.
– *****: El patrón que lo precede se repite 0 o mas veces.
– **?**: El patrón se repite 0 o 1 vez.
– **+**: El patrón se repite 1 o mas veces.
– **{x,y}**: El patrón se repite un mínimo de x veces y un máximo de y.
– **|**: Sirve para alternar expresiones, paracido al o lógico.
– Los corchetes **[]** incluidos en un patrón permiten especificar el rango de caracteres válidos a comparar.
– **[abc]** El patrón coincide con la cadena si en esta hay cualquiera de estos tres carácteres: a, b, c
– **[a-c]** Coincide si existe una letra en el rango ("a", "b" o "c")
– **c[ao]sa** Coincide con casa y con cosa
– **[^abc]** El patrón coincide con la cadena si en esta NO hay ninguno de estos tres carácteres: a, b, c. El signo ^ aquí tiene un valor excluyente
– **[0-9]** Coincide con una cadena que contenga cualquier número entre el 0 y el 9
– **()** Los paréntesis sirven para agrupar expresiones regulares. Ej.: (la|el)
– **|** Sirve para alternar expresiones. Ej.:(la|el) coincide si esta presente la o el.
– Para usar caracteres de otro lenguaje o caracteres especiales debemos "escapar el código", para ello deberemos usar la barra invertida '****'. Ej.: Si buscamos una cantidad en dólares (carácter usado por las expresiones regulares) 10$, pondríamos 10\$, ya que si escribiesemos 10$ buscaría un 10 a final de cadena, sería lo mismo con el símbolo del €.

Caracter	Coincidencia/Patrón	Ejemplo
^	Busca la coincicencia al inicio de la cadena	/^Esto/ coincidiría en: "Esto es una cadena"
$	Coincidencia con el final del input o del texto	/final$/ coincidiría con: "Esto es el final"
*	Coincide ninguna o muchas veces	/mi*/ coincidiría con: "miiiii" y tambien con "mi"
?	Coincide cero o una vez	/ap?/ coincidiría con: "apple"
+	Coincide una o muchas veces	/ap+/ coincidiría con: "apple"
{n}	Coincidiría exactamente n veces	/ap{2}/ coincidiría con: "apple" pero no con "apie"
{n,}	Coincidiría n o mas veces	/ap{2,}/ coincidiría con todas las p en: "apple" y "appple" pero no en "apie"
{n,m}	Coincidiría al menos n y como máximo m veces	/ap{2,4}/
.	Para cualquier caracter salvo salto de linea	/a.e/ coincide con "ape" y "axe"
[…]	Cualquier caracter dentro de los parentesis	/a[px]e/ coincidiría "ape" y "axe" pero no "ale"
[^…]	Cualquier caracter menos los que están entre paréntesis	/a[^px]/ coincidiría "ale" pero no "axe" o "ape"
b	Coincide con el primer límite de la palabra	/bno/ coincidiría con el primer "no" en la palabra "nono"
B	Coincide con el último límite de la palabra	/Bno/ coincide con el ultimo "no" en "nono"
d	Para dígitos de 0 a 9	/d{3}/ coincide 123 en "Now in 123"

Caracter	Coincidencia/Patrón	Ejemplo
D	Cualquier caracter no digito	/D{2,4}/ coincide con “Now ” en “Now in 123”
w	Cualquier caracter que sea una letra, digito o barra baja	
W	Lo contraio a lo anterior	/W/ Coincidiría “%” en “100%”
n	Para los saltos de linea	
s	Un solo espacio en blanco	
S	Un solo caracter pero que no sea un espacio en blanco	
t	Una tabulacion	
(x)	paréntesis de captura	Recuerda los caracteres coincidentes

3.1.3 Patrones

Patrón	Significado
.	cualquier carácter (excepto \n y \r)
^c	empezar por el carácter c
c$	terminar por el carácter c
c+	1 o más caracteres c
c*	0 o más caracteres c
c?	0 o 1 caracteres c
\n	nueva línea
\t	tabulador
\	escape, para escribir delante de caracteresespeciales: ^ . [ ] % ( ) | * ? { } \
(cd)	caracteres c y d agrupados
c|d	carácter c o d
c{n}	n veces el carácter c
c{n,}	n o más caracteres c
c{n,m}	desde n hasta m caracteres c
[a-z]	cualquier letra minúscula
[A-Z]	cualquier letra mayúscula
[0-9]	cualquier dígito
[cde]	cualquiera de los caracteres c, d o e
[c-f]	cualquier letra entre c y f (es decir, c, d, e o f)
[^c]	que no esté el carácter c
\w	cualquier letra o dígito o subrayado (pero no vocales acentuadas, ñ, ç, etc.)
\W	lo contrario de \w
\d	cualquier dígito
\D	lo contrario de \d
\s	cualquier espacio en blanco
\S	lo contrario de \s
\b	busca un emparejamiento a partir de un límite de palabra
\B	busca un emparejamiento cuando no es un límite depalabra

Modelos:

```
/^[a-zA-Z\s]+$/ // La cadena contiene contiene sólo caracteres (no dígitos) y
espacios

/[a-zA-Z\s]/    // La cadena contiene contiene algún caracter alfabético o espacio

/^[0-9]{3}\-[0-9]{3}\-[0-9]{3}$/ // Teléfono válido formato 976-306-100
```

```
/^[0-9]{3}\s[0-9]{3}\s[0-9]{3}$/ // Teléfono válido formato 976 306 100

/^[0-9]{3}\s[0-9]{2}\s[0-9]{2}\s[0-9]{2}$/ // Teléfono válido formato 976 30 61 00

/^\d{2}\/\d{2}\/\d{4}$/ // Fecha en formato 'dd/mm/yyyy'
/gratis/    // La cadena contiene 'gratis' en minúsculas.
/gratis/i  // La cadena contiene 'gratis', en mayúsculas o minúsculas
```

Ejemplo:

```
// Fecha en formato 'dd/mm/yyyy' (sólo se comprueba la estructura):
var patron01 = /^\d{2}\/\d{2}\/\d{4}$/;

// La cadena es exáctamente 'gratis' en minúsculas:
var patron02 = /^gratis$/;

// La cadena contiene 'gratis' en minúsculas:
var patron03 = /gratis/;

// La cadena contiene 'gratis', en mayúsculas o minúsculas:
var patron04 = /gratis/i;

// Sólo letras en mayúsculas/minúsculas, y espacios (OJO: no se incluyen 'ñ'
ni 'Ñ'):
var patron05 = /^[a-zA-Z\s]+$/;

// Sólo letras en mayúsculas/minúsculas, y espacios (se incluyen 'ñ' y 'Ñ'):
var patron06 = /^[a-zA-ZñÑ\s]+$/;

// Teléfono:
var patron07 = /^[0-9]{3}\-[0-9]{3}\-[0-9]{3}$/;

// DNI (sólo se comprueba la estructura, no si la letra es la adecuada):
var patron08 = /^[0-9]{2}\.[0-9]{3}\.[0-9]{3}\-[a-zA-Z]$/;

// Sólo números. No se admite cadena vacía:
var patron09 = /^[0-9]+$/;

// Sólo cadena vacía o números:
var patron10 = /^[0-9]*$/;

// Sólo letras en mayúsculas/minúsculas incluídas con tilde:
var patron11 = /^[a-zA-ZñÑáéíóúÁÉÍÓÚ]+$/;

// Sólo letras en mayúsculas/minúsculas, acentudadas, espacios y comillas
simples. No admite cadena vacía:
var patron12 = /^[a-zA-ZáéíóúÁÉÍÓÚäëïöüÄËÏÖÜàèìòùÀÈÌÒÙñÑ\s\']+$/;

// Sólo cadenas vacías, o con números y/o letras en mayúsculas/minúsculas
acentuadas,
// También espacios, comillas simples, dos puntos, puntos, comas, punto y coma
y guiones.:
var patron13 = /^[0-9a-zA-ZáéíóúÁÉÍÓÚäëïöüÄËÏÖÜàèìòùÀÈÌÒÙñÑ\s\'\:\.\,\;-]*$/;

// Sólo números y letras en mayúsculas/minúsculas acentuadas. No admite cadena
vacía:
var patron14 = /^[0-9a-zA-ZáéíóúÁÉÍÓÚäëïöüÄËÏÖÜàèìòùÀÈÌÒÙñÑ]+$/;

// Sólo números, letras en mayúsculas/minúsculas acentuadas, y espacios. No
admite cadena vacía:
var patron15 = /^[0-9a-zA-ZáéíóúÁÉÍÓÚäëïöüÄËÏÖÜàèìòùÀÈÌÒÙñÑ\s]+$/;

// E-Mail (sólo se comprueba la estructura):
```

```
var patron16 = /^[0-9a-z_\-\.]+@[0-9a-z\-\.]+\.[a-z]{2,4}$/i;
```

3.1.4 Modificadores

Los modificadores o "flags" son unas letras con un significado especial que se ponen detrás de la expresión regular, y matizan la forma de buscar.

- /a Explora una cadena hasta el final.
- **g** : /a/g ; Explora la cadena hasta el final, no se detiene con la primera coincidencia.
- **i** : /a/i ; No distingue entre mayúsculas y minúsculas.
- **m** : /a/m ; Permite usar varios ^ y $ en la cadena.
- **s** : /a/s ; Incluye el salto de línea en el comodín punto .
- **x** : /a/x ; ignora los espacios en el patrón .

Si escribimos más de un modificador en una expresión regular, debemos ponerlos en el mismo orden que aparecen arriba, es decir, en orden alfabético.

3.1.5 Formas de crear Expresiones Regulares

Si usamos una variable para guardar la expresión regular, ésta se convierte en un objeto del tipo RegExp, para ello usaremos:

```
expresion = / ...codigo_expresion .../

expresion = new RegExp(..codigo_expresion...).
```


Se pueden crear una Expresión Regular de dos formas:

Expresiones dinámicas:

```
/* Permite crear expresiones regulares en tiempo de ejecución */

var txt=new RegExp(patrón,attributos);
```


Expresiones estáticas:

```
/* Crea expresiones regulares estáticas, no se pueden modificar durante la
ejecución */

var txt=/patrón/attributos;
```

Si se usa la función constructor, la cadena de escape de reglas normal (precedida del caracter especial \ cuando incluye un string) es necesaria. Por ejemplo, lo siguiente es equivalente:

```
var re = new RegExp("\\w+");
var re = /\w+/;
```

Hay que remarcar que los parámetros en formato literal no usan comillas para indicar strings, mientras que los parámetros para la funcion constructor usan comillas. Así la siguiente expresión crea la misma expresión regular:

```
/ab+c/i;
new RegExp("ab+c", "i");
```

3.1.5.1 – Atributos del objeto RegExp.

Sirven para las dos formas de crear las expresiones:

Atributo	Significado
– m:	Si nuestra cadena contiene varias líneas físicas (`\n`) respeta esos saltos de línea, lo que significa, por ejemplo, que las anclas `^ $` no se aplican al principio y final de la cadena, sino al principio y final de cada línea.
– i:	Se confronta el patrón con la cadena ignorando mayusculas y minúsculas.
– g:	Realiza una búsqueda global, no se detiene en la primera ocurrencia que encuentra.

3.1.5.2 – Ejecución y métodos del objeto RegExp

Para ejecutar las expresiones regulares disponemos de métodos, el método test():

resultado = expresionRegular.test("texto a comprobar")

En la variable `resultado` se guardará un valor booleano `true` si la cadena contiene la expresión al menos una vez, y `false` si no la contiene.


```
texto1 = "Los martes voy a clase.";
texto2 = "Los miércoles también voy a clase.";
buscar = /martes/;
alert(buscar.test(texto1)); //devuelve true
alert(buscar.test(texto2)); //devuelve false
```

```
texto1 = "a = 4.65";
texto2 = "b = 17.3";
buscar = /[aeiou]/;
alert(buscar.test(texto1)); //devuelve true
alert(buscar.test(texto2)); //devuelve false
```



```
texto1 = "El profesor tiene 67 años";
texto2 = "El profesor tiene 26 años";
buscar = /[1-5]/;
alert(buscar.test(texto1)); //devuelve false
alert(buscar.test(texto2)); //devuelve true
```

```
texto1 = " El profesor cumple 20 años";
texto2 = " El profesor cumple veinte años";
buscar = /\d/;
alert(buscar.test(texto1)); //devuelve true
alert(buscar.test(texto2)); //devuelve false
```

```
texto1 = "Hoy es lunes";
texto2 = " ";
buscar = /\w/;
alert(buscar.test(texto1)); //devuelve true
alert(buscar.test(texto2)); //devuelve false
```

```
texto1 = "Hoy es lunes";
texto2 = "";
```

```
buscar = /./;
alert(buscar.test(texto1)); //devuelve true
alert(buscar.test(texto2)); //devuelve false
```

```
texto1 = "cole";
texto2 = "Mi cole es grande";
buscar = /^cole$/;
alert(buscar.test(texto1)); //devuelve true
alert(buscar.test(texto2)); //devuelve false
```

Método	**Significado**
– compile():	El método compile convierte el patrón en un formato interno para que la ejecución sea más rápida. Por ejemplo, esto permite un uso más eficiente de expresiones regulares en bucles → `compilado = patron.compile(patron);` La variable compilado guardará la expresion regular ya compilada.
– exec():	busca la expresión en el texto, y devuelve el primer texto que concuerda con la expresión buscada. Si no encuentra ninguna coincidencia, entonces devuleve null. → `buscar = patron.exec(texto);` Este método tiene además dos propiedades, la propiedad `.index`, la cual nos indica la posición en la que se encuentra la cadena buscada, y la propiedad `.input`, la cual devuelve la cadena completa en la que estamos realizando la búsqueda. La posición que indica la propiedad index es la del primer caracter encontrado en el trozo de cadena que coincide. este se empieza a contar a partir del 0. → `posicion = patron.exec(texto).index; textoCompleto = patron.exec(texto).input`
– test():	Busca el patrón en el texto y devuelve el valor booleano true si el texto se ajusta, si no es así devolverá false. → `buscar = patron.test("cadena")`

3.1.5.3 - Propiedades del objeto RegExp

Las propiedades se usan con el objeto RegExp y no con cualquier expresión regular concreta, su sintaxis es de la forma: RegExp.propiedad; son de sólo lectura, por lo que no las podemos modificar directamente, sin embargo se actualizan automáticamente cada vez que se emplea un método con un objeto regular, ya sea con los métodos de RegExp o con los métodos de String; correspondiendo la cadena explorada a la obtenida tras el último método empleado.

Las propiedades son:

$1 ... $9 → Índices que contienen las partes agrupadas con paréntesis en el patrón de búsqueda.

Input → RegExp.input: Cadena que se ha explorado.

lastMatch → RegExp.lastMatch: Última coincidencia encontrada.

Multiline → RegExp.multiline: Variable booleana que indica si la cadena explorada incluye saltos de línea.

lastParent → RegExp.lastParent: Última coincidencia encontrada con un patrón entre paréntesis.

leftContext → RegExp.leftContext: Desde el principio de la cadena hasta la coincidencia hallada.

rightContext → RegExp.rightContext: Desde la coincidencia hasta el final de la cadena.

3.1.6 - Ejemplos

Validar una matrícula de coche:

```
function validaMatricula() {
  var mat = document.getElementById("matricula").value;
  var ex1 = new RegExp("^[0-9]{4} [A-Z]{3}$");
  var ex2 = /^[0-9]{4} [A-Z]{3}$/;
  if(ex1.test(mat))
    alert("Ok");
  else
    alert("Error");
  if(ex2.test(mat))
    alert("Ok");
  else
    alert("Error");
}
```

Validar una fecha (sólo el formato):

```
Function validaFecha() {
  var fech = document.getElementById("fecha").value;
  var ex1 = new RegExp("^(0?[1-9]|[12][0-9]|3[01])\/(0?[1-9]|1[012])\/[0-9]{1,2}$");
  var ex2 = /^(0?[1-9]|[12][0-9]|3[01])\/(0?[1-9]|1[012])\/[0-9]{1,2}$/;
  if(ex1.test(fech))
   alert("Ok");
  else
   alert("Error");
  if(ex2.test(fech))
   alert("Ok");
  else
   alert("Error");
}
```

Ejemplo exec():

```
> re = /d(b+)(d)/ig
/d(b+)(d)/gi
> z = "dBdxdbbdzdbd"
'dBdxdbbdzdbd'
> resultado = re.exec(z)
[ 'dBd', 'B', 'd', index: 0, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
3
> resultado = re.exec(z)
[ 'dbbd', 'bb', 'd', index: 4, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
8
> resultado = re.exec(z)
[ 'dbd', 'b', 'd', index: 9, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
12
> z.length
12
> resultado = re.exec(z)
Null
```

```
>> re = /d(b+)(d)/ig
<- /d(b+)(d)/gi
>> z = "dBdxdbbdzdbd"
<- "dBdxdbbdzdbd"
>> resultado = re.exec(z)
<- Array(3) [ "dbbd", "bb", "d" ]
     0: "dbbd"
     1: "bb"
     2: "d"
     groups: undefined
     index: 4
     input: "dBdxdbbdzdbd"
     length: 3
   ▶ <prototype>: Array []
```

3.2.- 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
A4
```

Podremos emplear las comillas dobles o simples:

```
var cadena = '<input type="checkbox" name="coche" />Audi A4';
var cadena = "<input type='checkbox' name='coche' />Audi A4";
```

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 A4";
```

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.
\'	Comilla simple.
\\	Barra inclinada.
\b	Retroceso.
\t	Tabulador.
\n	Nueva línea.
\r	Salto de línea.
\f	Avance de página.

3.2.1.- 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
length	Contiene la longitud de una cadena.

Métodos del objeto String

Métodos	Descripción
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.
fromCharCode()	Convierte valores Unicode a caracteres.
indexOf()	Devuelve la posición de la primera ocurrencia del carácter buscado en la cadena.
lastIndexOf()	Devuelve la posición de la última ocurrencia del carácter buscado en la cadena.
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 cadena especificada.
search()	Busca una subcadena en la cadena y devuelve la posición dónde se encontró.
slice()	Extrae una parte de la cadena y devuelve una nueva 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 especificados.
toLowerCase()	Convierte una cadena en minúsculas.
toUpperCase()	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/>");
```

Ejemplo Extraer de una cadena de texto todas aquellas palabras que tengan la combinación de letras "**ía**"

```
texto = 'que en su día me parecieron confusos, incluso corruptos —como les pareció y parece a muchos
editores—, hoy son para mí claros, al punto de defender que han de dejarse como están, y por otro lado hay
pasajes que hoy ya no he dudado en enmendar, por más que la alteración sea de tal calibre que produzca el
rechazo de algún especialista. Y es que no basta estar familiarizado con los usos y abusos sintácticos del
alcalaíno y sus contemporáneos: también con los procedimientos y artimañas de los operarios de las imprentas
de aquel tiempo. Hay quien cree en los duendes de las imprentas: yo ni afirmo ni niego, sólo digo que los
operarios de aquéllas se bastaban y sobraban para todo tipo de travesuras: algunas, burdas, y otras, casi
indetectables; pero alguna parte de responsabilidad correspondería a los autores, pues los resultados finales
evidencian que no siempre la imprenta recibía un manuscrito impecable, bien ordenado y con buena caligrafía,
que facilitase la composición del libro, y el Quijote es el mejor ejemplo de ello. Por otro lado Cervantes,
como otros de nuestros clásicos castellanos, gustaba de salpicar el texto con composiciones poéticas y se las
ingeniaba para poner en boca de algún personaje alguna que otra novelita intermedia de corte amoroso,
aventurero o picaresco; y más aun: a veces no podía resistirse a discursear sobre algún asunto político-social
o literario que le inquietase. Todo ello afecta a la fluidez en la lectura del Quijote, en especial la Primera
Parte, donde los protagonistas desaparecen durante páginas y páginas y se observan clamorosos lapsus en el
hilo argumental, resultantes de inserciones efectuadas a posteriori y por aplicación manu militari de la ley
del encaje. Los editores no solemos arreglar esos descalabros, porque supondría demasiada intervención en el
texto, y así, esta edición electrónica se ciñe a lo acostumbrado. La ortografía se ha actualizado, pero no al
extremo que restase encanto a un texto de cuatrocientos años, y se justifica en las notas la menor alteración
aplicada al texto primitivo.'

er = new RegExp("\\w+ía","g")

texto.match(er)
```

Comentado [D1]:

Resultado:

```
Array(7) [ "día", "correspondería", "recibía", "caligrafía", "podía", "supondría", "ortografía" ]
0: "día"
1: "correspondería"
2: "recibía"
3: "caligrafía"
4: "podía"
5: "supondría"
6: "ortografía"
length: 7
<prototype>: Array []
```

3.3.- 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.
```

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

Método	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(y,x)`	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 neperiando (base E) de x.
`max(x,y,z,...,n)`	Devuelve el número más alto de los que se pasan como parámetros.
`min(x,y,z,...,n)`	Devuelve el número más bajo de los que se pasan como parámetros.
`pow(x,y)`	Devuelve el 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 está 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 />");
```

3.4.- 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.

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

Propiedades del objeto Number

Propiedad	Descripción
`constructor`	Devuelve la función que creó el objeto `Number`.
`MAX_VALUE`	Devuelve el número más alto disponible en JavaScript.
`MIN_VALUE`	Devuelve el número más pequeño disponible en JavaScript.
`NEGATIVE_INFINITY`	Representa a infinito negativo (se devuelve en caso de `overflow`).
`POSITIVE_INFINITY`	Representa a infinito positivo (se devuelve en caso de `overflow`).
`prototype`	Permite añadir nuestras propias propiedades y métodos a un objeto.

Métodos del objeto Number

Método	Descripcción
`toExponential(x)`	Convierte un número a su notación exponencial.
`toFixed(x)`	Formatea un número con x digitos decimales después del punto decimal.

`toPrecision(x)`	Formatea un número a la longitud 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:

```
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);
```

3.5.- Objeto Boolean.

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

Popiedad	Descripción
`constructor`	Devuelve la función que creó el objeto `Boolean`.
`prototype`	Te permitirá añadir propiedades y métodos a un objeto.

Método	Descripción
`toString()`	Convierte un valor `Boolean` a una cadena y devuelve el resultado.
`valueOf()`	Devuelve el valor primitivo de un objeto `Boolean`.

Ejemplos:

```
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.

Ejemplo del funcionamiento de este objeto.

```
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
```

3.6.- Objeto Date.

El objeto **`Date`** se utiliza para trabajar con fechas y horas. Los objetos **`Date`** se crean con **`new Date()`**.

Hay 4 formas de instanciar (crear 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.)
```

Propiedades del objeto Date

Propiedad	Descripción
`constructor`	Devuelve la función que creó el objeto `Date`.
`prototype`	Te permitirá añadir propiedades y métodos a un objeto.

Métodos del objeto Date

Método	Descripción
`getDate()`	Devuelve el día del mes (de 1-31).
`getDay()`	Devuelve el día de la semana (de 0-6).
`getFullYear()`	Devuelve el año (4 dígitos).
`getHours()`	Devuelve la hora (de 0-23).
`getMilliseconds()`	Devuelve los milisegundos (de 0-999).
`getMinutes()`	Devuelve los minutos (de 0-59).
`getMonth()`	Devuelve el mes (de 0-11).
`getSeconds()`	Devuelve los segundos (de 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 (de 1-31).
`getUTCDay()`	Devuelve el día de la semana en base a la hora UTC (de 0-6).
`getUTCFullYear()`	Devuelve el año en base a la hora UTC (4 dígitos).
`setDate()`	Ajusta el día del mes del objeto (de 1-31).
`setFullYear()`	Ajusta el año del objeto (4 dígitos).
`setHours()`	Ajusta la hora del objeto (de 0-23).

Ejemplos :

```
 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);
```

3.7 Elemento canvas

El elemento **<canvas>** se usa para dibujar gráficos 2D y3D con aceleración por hardware en páginas web. Las aplicaciones de Mozilla soportan **<canvas>** desde Firefox 1.5. El elemento fue inicialmenmte presentado por Apple para el Dashboard de OS X y Safari. Internet Explorer soporta **<canvas>** desde la versión 9 en adelante; para versiones anteriores de IE, se puede añadir soporte para **<canvas>** a una página incluyendo un script del proyecto de Google Explorer Canvas. Google Chrome y Opera 9 también soportan.

3.7.1 Etiqueta canvas

El elemento `<canvas>` es parecido al elemento `<img>`, con la diferencia que este no tiene los atributos src y alt. El elemento `<canvas>` tiene solo dos atributos `width` y `height`. Ambos son opcionales y pueden ser definidos usando propiedades DOM. Cuando los atributos ancho y alto no estan especificados, el lienzo se inicializara con 300 pixels ancho y 150 pixels de alto. El elemento puede ser arbitrariamente redimensionado por CSS, pero durante el renderizado la imagen es escalada para ajustarse al tamaño de su layout. Si el tamaño del CSS no respeta el ratio del canvas inicial, este aparecerá distorsionado.

```
<canvas id="micanvas" width="150" height="150">
  Texto dentro el camvas
  <img src="img/imagen.png" width="150"
height="150" alt=""/>
</canvas>
```

3.7.2 Contenido o contexto de renderización

La etiqueta <canvas> crea un lienzo de dibujo fijado que expone uno o mas contextos renderizados, los cuales son usados para crear y manipular el contenido mostrado. Inicialmente la renderizacion es en contextos 2D. Otros contextos deberan proveer diferentes tipos de renderizaciones; por ejemplo, WebGL usa un 3D contexto ("experimental-webgl") basado sobre OpenGL ES.

El canvas esta inicialmente en blanco. Para mostrar algun contenido en el, primero se necesita acceder al contexto a renderizar y después dibujar sobre este. El elemento `<canvas>` tiene un metodo llamado `getContext()`, usado para obtener el contexto a renderizar y sus funciones de dibujo. `getContext()` toma un parametro, el tipo de contexto para graficos 2D, es "2d".


```
var canvas = document.getElementById('micanvas');
var contenido = canvas.getContext('2d');
```

3.7.3 Comproción de interpretación para canvas.

El contenido que se inserta entre las etiquetas <canvas> … </canvas> es mostrado en navegadores los cuales no soportan <canvas>. Puede comprobarse desde programacion por un simple bucle:

```
var canvas = document.getElementById('micanvas');

if (canvas.getContext){
  var contenido = canvas.getContext('2d');
  // código de dibujo
} else {
  // sin soporte canvas
}
```

3.7.4 Ejemplo canvas sencillo.

Para generar

```
<html>
 <head>
  <script type="application/javascript">
    function dibuja() {
      var canvas = document.getElementById("micanvas");
      if (canvas.getContext) {
        var contenido = canvas.getContext("2d");

        contenido.fillStyle = "rgb(200,0,0)";
        contenido.fillRect (10, 10, 55, 50);

        contenido.fillStyle = "rgba(0, 0, 200, 0.5)"; //con parámetro alfa
        contenido.fillRect (30, 30, 55, 50);
      }
    }
  </script>
 </head>
 <body onload="dibuja();">
   <canvas id="micanvas" width="150" height="150"></canvas>
 </body>
</html>
```

3.7.5 Métodos del objeto canvas.

MÉTODO	USO
Aplica el strokeStyle (linéa) traza la figura diseñada con las herramientas moveTo y lineTo	.stroke()
crea una nueva ruta (dibujo)	.beginPath()
desplazamos a una posición determinada. Es como desplazar un lapiz hasta un punto.	.moveTo(x,y)
desplazamos el lápiz a otro punto, dibujando una línea por el camino	.lineTo(x,y)
estilo de línea	.strokeStyle(argumentos)
Grosor de la línea	.lineWidth = x
Estilo borde línea	. lineCap = "butt|round|square"
Estilo unión líneas	.lineJoin = "bevel|round|miter";
rectángulos	.strokeRect(x, y, ancho, alto)
estilo de rectángulo	.fillRect(x, y, ancho, alto)
rectángulo eliminar	.clearRect(x, y, ancho, alto)
figura rellena	.fill()
arcos	.arc(x, y, radius, ángulo inicio, ángulo fin, sentido)
curvas de Bézier	.bezierCurveTo(p1x, p1y, p2x, p2y, x, y)
curvas cuadráticas	.quadraticCurveTo(px, py, x, y)

textos	.font(Font) .fillText(texto, x, y) .strokeText(texto, x, y) .textAlign(start|end|left|right|center) .measureText()
pintar imágenes	.drawImage(objeto, x, y) .drawImage(objeto, x, y, ancho, alto) .drawImage(objeto, x1, y1, ancho1, alto1, x2, y2, ancho2, alto2)
transparencias	.fillStyle = 'rgb(r,g,b,alpha)'
sombras	.shadowOffsetX = valor .shadowOffsetY = valor .shadowColor = 'rgb(r,g,b,alpha)' .shadowBlur = valor
gradiente lineal	.createLinearGradient(x1, y1, x2, y2) .addColorStop(punto, color)
gradiente radial	.createRadialGradient(x1, y1, radioInicio, x2, y2, radioFinal) .addColorStop(punto, color)
patrón imagen	.createPattern(imagen, repetición) repeticion = repeat|repeat-x|repeat-y|no-repeat
patrón canvas	.createPattern(canvas, repetición) repeticion = repeat|repeat-x|repeat-y|no-repeat
guardar y restaurar	.save() .restore()
trasladar	.translate(x, y)
rotar	.rotate(grados)
redimensionar	.scale(x, y)

CANVAS LINEAS

```
lienzo.strokeStyle = "rgb(200,0,0)";
lienzo.beginPath();
lienzo.moveTo(0, 0);
lienzo.lineTo(150, 300);
lienzo.lineTo(300, 0);
lienzo.stroke();
```


CANVAS ESTILO DE LÍNEA

```
lienzo.strokeStyle = "rgb(200,0,0)";
lienzo.lineWidth = 7; //grosor de la línea
lienzo.beginPath();
lienzo.lineCap = "butt";
lienzo.lineJoin = "bevel"; //estilo unión línea
lienzo.moveTo(20, 20);
lienzo.lineTo(150, 300);
lienzo.lineTo(300, 50);
lienzo.stroke();
```

CANVAS RECTANGULO

```
lienzo.strokeStyle = "rgb(200,0,0)"; //RECTANGULO VACIO
lienzo.strokeRect(50, 100, 30, 60); //RECTANGULO VACIO

lienzo.fillStyle = "rgb(255,0,0)"; //RECTANGULO RELLENO
lienzo.fillRect(250, 250, 80, 110); //RECTANGULO RELLENO

lienzo.fillStyle = "rgb(0,200,0,0.3)"; //RECTANGULO RELLENO Y CON TRANSPARENCIA
lienzo.fillRect(275, 220, 30, 60); //RECTANGULO RELLENO
```

CANVAS BORRADO RECTANGULO

```
lienzo.fillStyle = "rgb(200,100,0)";
lienzo.fillRect(30, 30, 300, 300);
lienzo.clearRect(100, 100, 20, 20); //ELIMINA PATTE DEL RELLENO UN RECTÁNGULO
lienzo.clearRect(240, 100, 20, 20);
lienzo.clearRect(140, 200, 80, 30);
```

CANVAS FIGURA RELLENA

```
lienzo.strokeStyle = "red";
lienzo.fillStyle = "yellow "; //RELLENO FIGURA
lienzo.lineWidth = 5; //ANCHURA DE LA LÍNEA
lienzo.beginPath();
lienzo.moveTo(150, 10);
lienzo.lineTo(10, 290);
lienzo.lineTo(290, 290);
lienzo.lineTo(150, 10);
lienzo.fill();
lienzo.stroke();
```

CANVAS ARCOS + ARCOS RELLENOS

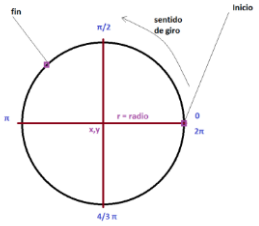
```
lienzo.lineWidth = 5;

lienzo.strokeStyle = "red";
lienzo.beginPath();
lienzo.arc(100, 100, 50, 0, Math.PI, true); //DIBUJA UN ARCO
lienzo.stroke();

lienzo.strokeStyle = "blue";
lienzo.beginPath();
lienzo.arc(100, 200, 50,0, Math.PI, false);
lienzo.stroke();

lienzo.strokeStyle = "yellow";
lienzo.beginPath();
lienzo.arc(200, 100, 50, 0, Math.PI, true);
lienzo.stroke();

lienzo.fillStyle = "green"; //DIBUJARA EL ARCO RELLENO
lienzo.beginPath();
lienzo.arc(200, 200, 50, 0, Math.PI, true);
lienzo.fill();
```



CANVAS CURVAS DE BEZIER

```
//CURVA
lienzo.lineWidth = 5;
lienzo.strokeStyle = "red";
lienzo.beginPath();
lienzo.moveTo(0, 150);
lienzo.bezierCurveTo(100, 50, 200, 250, 350, 150); //Línea Curva Bézier
lienzo.stroke();
```

```
//PUNTO A
lienzo.strokeStyle = "green";
lienzo.fillStyle = "green";
lienzo.lineWidth = 1;

lienzo.strokeText("p A", 105, 50);
lienzo.beginPath();
lienzo.arc(100, 50, 3, 0, 2 * Math.PI, true);
lienzo.fill();

//punto B
lienzo.strokeText("p B", 205, 250);
lienzo.beginPath();
lienzo.arc(200, 250, 3, 0, 2 * Math.PI, true);
lienzo.fill();
```

CANVAS CURVAS CUADRÁTICAS

```
<script>
    var fila = 0;
    var direccion = true;

    function devolverLienzo(x){
        return  (document.getElementById(x)).getContext("2d");
    }

    function dibujar() {
        var lienzo = devolverLienzo("micanvas");
        if (lienzo) {
        lienzo.clearRect(0, 0, 350, 350);
        lienzo.lineWidth = 5;
        lienzo.strokeStyle = "red";
        lienzo.beginPath();
        lienzo.moveTo(0, 150);
        lienzo.quadraticCurveTo(fila, fila, 300, 150);
        lienzo.stroke();
        lienzo.strokeStyle = "green";
        lienzo.fillStyle = "green";
        lienzo.lineWidth = 1;
        lienzo.beginPath();
        lienzo.moveTo(fila,fila);
        lienzo.arc(fila, fila, 2, 0, 2 * Math.PI);
        lienzo.fill();
        lienzo.strokeText("p", fila - 2, fila - 7);
        lienzo.strokeText(fila + ", "+ fila,10,240);

        if(direccion) {
            fila++;
        }
        else{
            fila--;
        }
        if (fila > 349 || fila < 1) {
            direccion = ! direccion;
        }
        }
    }
    window.onload = function iniciar() {
        setInterval(dibujar, 20);}
</script>
```

CANVAS CURVAS CUADRÁTICAS CON REINICIO (ejercio complerto)

```
<!DOCTYPE html>
        <html lang="es">
        <head>
            <meta charset="UTF-8">
            <title>CANVAS</title>
            <style>
                #micanvas{
                    border: #92C428 solid 3px;
                }
            </style>
            <script>
                var fila = 0;
                var direccion = true;
                var movimiento;
                function c() {
                    var canvas = document.getElementById('micanvas');
                    var contenido = canvas.getContext('2d');
                    if (contenido) {
                        contenido.clearRect(0, 0, 370, 370);
                        contenido.lineWidth = 5;
                        contenido.strokeStyle = "red";
                        contenido.beginPath();
                        contenido.moveTo(0, 150);
                        contenido.quadraticCurveTo(fila, fila, 300, 150);
                        //contenido.quadraticCurveTo(450, 450, 300, 150);
                        contenido.stroke();
                        contenido.strokeStyle = "green";
                        contenido.fillStyle = "green";
                        contenido.lineWidth = 1;
                        contenido.beginPath();
                        contenido.moveTo(fila,fila);
                        contenido.arc(fila, fila, 2, 0, 2 * Math.PI);
                        contenido.fill();
                        contenido.strokeText("p", fila - 2, fila - 7);
                        contenido.strokeText(fila + ", "+ fila,10,240);

                        if(direccion) {
                            fila++;
                        }
                        else{
                            fila--;
                        }
                        if (fila > 349 || fila < 1) {
                            direccion = ! direccion;
                        }
                    }


                }
                function inicio(){
                    movimiento = window.setInterval(c,20);
                    document.getElementById("miBoton").value = " Parar ";
                    document.getElementById("miBoton").onclick = parar;
                }
                function parar(){
                    window.clearInterval(movimiento);
                    document.getElementById("miBoton").value = "Iniciar";
                    document.getElementById("miBoton").onclick = inicio;
                }
                window.onload = function (){
                    document.getElementById("miBoton").onclick = inicio;
                }
            </script>
```

```
        </head>
<body>
<form>
    <input type="button" id="miBoton" value="Iniciar">
</form>
<br>
<canvas id="micanvas" width="500" height="500">
</canvas>
</body>
</html>
```

CANVAS TEXTOS

```
//texto negrita de 25 pixeles y letra Arial
lienzo.font = "bold 25px Arial";
lienzo.fillStyle = "green";
lienzo.strokeStyle = "red";
lienzo.textAlign = "center";
lienzo.fillText("texto con relleno verde", 175, 150);
//mesaureText --> devuelve la longitud del texto
var ancho = lienzo.measureText("texto con relleno verde");
lienzo.textAlign = "center";
//toPrecision(x) --> indica la cantidad de dígitos del número.
lienzo.strokeText("ancho=" + ancho.width.toPrecision(6) + "px", 175, 250 );

//createLinearGradient --> crea un gradiente lineal desde un punto hasta otro
var gradiente=lienzo.createLinearGradient(0,0,250,0);
gradiente.addColorStop("0.2","yellow");
gradiente.addColorStop("0.5","red");
gradiente.addColorStop("0.8","blue");
gradiente.addColorStop("1.0","pink");
lienzo.strokeStyle=gradiente;
lienzo.strokeText("Texto con gradiente",170,350);
```

CANVAS PINTAR IMAGENES

```
            lienzo.fillStyle = "rgb(0,0,200)";
            lienzo.lineWidth = 7;
            lienzo.fillRect(0, 0, 600, 600);
            img1 = new Image();
            img1.src = "logo.jpg"; //imagen de dimensiones de 77px por 57px
            img1.onload = function () {
            //lienzo.drawImage(imagen, xOrigen, yOrigen, OrigenAncho,
OrigenAlto, xDestino, yDestino, DestinoAlto, DestinoAncho)
            lienzo.drawImage(img1, 150, 0);
            lienzo.drawImage(img1, 300, 150, 100, 50);
            lienzo.drawImage(img1, 0, 0, 77, 27, 100, 350, 72, 72);
```

CANVAS TRANSPARENCIAS

```
//sin transparencia
lienzo.fillStyle = "rgb(200,200,0)";
lienzo.fillRect(10, 10, 250, 250);
//con una transparencia del 50%
lienzo.fillStyle = "rgba(200,200,0,0.5)";
lienzo.fillRect(200, 200, 250, 250);
```

CANVAS SOMBRAS

```
lienzo.shadowOffsetX = -10;
lienzo.shadowOffsetY = 5;
lienzo.shadowBlur = 10; //longitud del degradado de la sombra
lienzo.shadowColor = "rgba(0,100,100,0.5)";
```

```
lienzo.fillStyle = "rgb(0,100,100)";
lienzo.fillRect(20, 20, 150, 100);

lienzo.shadowOffsetY = 15;
lienzo.shadowOffsetX = 15;
lienzo.shadowBlur = 20;
lienzo.shadowColor = "rgba(0,050,050,0.5)";
lienzo.fillStyle = "rgb(0,150,150)";
lienzo.fillRect(300, 120, 100, 100);
```

CANVAS GRADIENTE LINEAL

```
var gradiente1 = lienzo.createLinearGradient(0, 0, 500, 0);
gradiente1.addColorStop(0.05, "red");
gradiente1.addColorStop(0.20, "orange");
gradiente1.addColorStop(0.35, "yellow");
gradiente1.addColorStop(0.50, "green");
gradiente1.addColorStop(0.65, "blue");
gradiente1.addColorStop(0.80, "darkblue");
gradiente1.addColorStop(0.95, "violet");
lienzo.fillStyle = gradiente1;
lienzo.fillRect(0, 0, 520, 200);
```

CANVAS GRADIENTE RADIAL

```
//createRadialGradient(x0,y0,r0,x1,y1,r1) --> x0,y0,r0 origen del círculo 1 y radio y x1,y1,r1
origen del círculo 2 y radio
var gradiente1 = lienzo.createRadialGradient(110, 200, 0, 110, 200, 200);
//addColorStop(fin,color)  fin --> Un valor entre 0,0 y 1,0 que representa la posición entre
el inicio y el final en un gradiente.
gradiente1.addColorStop(0, "blue");
gradiente1.addColorStop(0.5, "red");
lienzo.fillStyle = gradiente1;
//arc(x, y, centro, ang_ini, ang_fin, sentido)
/* x e son las coordenadas del centro del arco, el radio del arco los ángulos inicial y final
(en qué ángulo empezar y terminar de dibujar el círculo, en radianes)
la dirección hacia la que se dibujará (false para seguir el sentido de las agujas del reloj,
que es el valor por defecto, o true para el sentido contrario). Este parámetro es opcional*/
lienzo.arc(110, 200, 100, 0, Math.PI*2, true);
lienzo.fill();
```

CANVAS PATRON DE IMAGEN

```
img = new Image();
img.src = "logo.jpg";
img.onload = function () {
      var patron = lienzo.createPattern(img, "repeat");
      // Opciones del patón --> repeat, repeat-x, repeat-y, no-repeat
      lienzo.fillStyle = patron;
      lienzo.fillRect(0, 0, 600, 600);
        }
```

CANVAS - PATRON MEDIANTE OTRO CANVAS

```
lienzo1.strokeStyle = "blue ";
lienzo1.beginPath();
lienzo1.moveTo(45, 5);
lienzo1.lineTo(5, 45);
lienzo1.lineTo(45, 45);
lienzo1.lineTo(5, 5);
lienzo1.lineTo(45, 5);
lienzo1.stroke();
```

```
//.createPattern(canvas, repetición) --> repeticion = repeat|repeat-x|repeat-y|no-repeat
var patron = lienzo2.createPattern(lienzo1.canvas, "repeat");
lienzo2.fillStyle = patron;
lienzo2.fillRect(0, 0, 600, 600);
```

CANVAS GUARDAR Y RESTAURAR

```
lienzo.fillStyle = "greenyellow";
lienzo.fillRect(10, 30, 100, 100);
lienzo.save(); //graba propiedades del lienzo actual
micuadradoGradiente(lienzo, 120, 30, 100, 100); //aplico un nuevo fillStyle con la
función
lienzo.fillRect(230, 30, 100, 100);
lienzo.restore(); // reutiliza las propiedades grabadas
lienzo.fillRect(230, 150, 100, 100);
```

CANVAS TRASLADAR

```
lienzo.save();
lienzo.translate(x, y);
lienzo.fillStyle = "orange";
lienzo.beginPath(); // Crea un nuevo trazo, los comandos de dibujo futuros son aplicados
dentro del trazo y usados para construir el nuevo trazo hacia arriba.
lienzo.moveTo(base / 2, 0);
lienzo.lineTo(0, altura);
lienzo.lineTo(base, altura);
lienzo.lineTo(base / 2, 0);
lienzo.fill();
lienzo.restore();
```

CANVAS ROTAR

```
lienzo.clearRect(0, 0, 350, 350);
lienzo.save();
lienzo.fillStyle = "orange";
lienzo.translate(170, 165);
lienzo.rotate(avance); //angulo de giro del lienzo
lienzo.fillRect(-100, -100, 200, 200);
lienzo.restore();
avance += 0.01;
if (avance > Math.PI * 2) {
   avance = 0;
}
```


CANVAS REDIMENSIONAR

```
lienzo.clearRect(0, 0, 800, 800);
lienzo.save();
lienzo.translate(400, 400);
lienzo.rotate(giro);
lienzo.scale(tamanoX, tamanoY);
lienzo.drawImage(img1, -125, -125);
giro += 0.075;
tamanoX += 0.02;
tamanoY += 0.02;

if (giro > Math.PI * 2) {
   giro = 0;
}
```

```
if (tamanoX >= 10) {
   tamanoX = 0.01;
   tamanoY = 0.01;
}

lienzo.restore();
```

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

`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 partirde 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)`. Evalua 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)`. Evalua 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.

Existen otras 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 y que animo a investigarlas sobre el objeto window.

y cuya explicación remito como ejercicio al lector interesado en saber más sobre el objeto window.

Ejercicio

```
<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.

Ejercicio

```
<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.

```
<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";
```

Más información:http://www.w3schools.com/js/js_obj_string.asp

Propiedades del objeto String

Propiedad	Descripción
`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

Propiedades del objeto String

Método	Descripción
`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 indices 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

Method	Description
`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

ANEXO V - Relojes, contadores e intervalos de tiempo

En ocasiones, algunas páginas web muestran un reloj con la hora actual. Si el reloj debe actualizarse cada segundo, no se puede mostrar la hora directamente en la página HTML generada por el servidor. En este caso, aunque existen alternativas realizadas con Java y con Flash, la forma más sencilla de hacerlo es mostrar la hora del ordenador del usuario mediante JavaScript.

Para crear y mostrar un reloj con JavaScript, se debe utilizar el objeto interno `Date()` para crear fechas/horas y las utilidades que permiten definir contadores, para actualizar el reloj cada segundo.

El objeto `Date()` es una utilidad que proporciona JavaScript para crear fechas y horas. Una vez creado un objeto de tipo fecha, es posible manipularlo para obtener información o realizar cálculos con las fechas. Para obtener la fecha y hora actuales, solamente es necesario crear un objeto `Date()` sin pasar ningún parámetro:


```
var fechaHora = new Date();
```

Utilizando el código anterior, se puede construir un reloj muy básico que no actualiza su contenido:


```
var fechaHora = new Date();
document.getElementById("reloj").innerHTML = fechaHora;

<div id="reloj" />
```

Cuando se carga la página, el ejemplo anterior mostraría un texto parecido al siguiente en el `<div>` reservado para el reloj:

```
Mon May 04 2009 13:36:10 GMT+0200 (Hora de verano romance)
```

Este primer reloj construido presenta muchas diferencias respecto al reloj que se quiere construir. En primer lugar, muestra más información de la necesaria. Además, su valor no se actualiza cada segundo, por lo que no es un reloj muy práctico.

El objeto `Date()` proporciona unas funciones muy útiles para obtener información sobre la fecha y la hora. Concretamente, existen funciones que devuelven la hora, los minutos y los segundos:

```
var fechaHora = new Date();
```

```
var horas = fechaHora.getHours();
var minutos = fechaHora.getMinutes();
var segundos = fechaHora.getSeconds();

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

<div id="reloj" />
```

Utilizando las funciones `getHours()`, `getMinutes()` y `getSeconds()` del objeto `Date`, el reloj solamente muestra la información de la hora. El resultado del ejemplo anterior sería un reloj como el siguiente:

```
20:9:21
```

Si la hora, minuto o segundo son menores que `10`, JavaScript no añade el `0` por delante, por lo que el resultado no es del todo satisfactorio. El siguiente código soluciona este problema añadiendo un `0` cuando sea necesario:

```
var fechaHora = new Date();
var horas = fechaHora.getHours();
var minutos = fechaHora.getMinutes();
var segundos = fechaHora.getSeconds();

if(horas < 10) { horas = '0' + horas; }
if(minutos < 10) { minutos = '0' + minutos; }
if(segundos < 10) { segundos = '0' + segundos; }

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

<div id="reloj" />
```

Ahora el reloj muestra correctamente la hora:

```
20:14:03
```


Si se quiere mostrar el reloj con un formato de 12 horas en vez de 24, se puede utilizar el siguiente código:

```
var fechaHora = new Date();
var horas = fechaHora.getHours();
var minutos = fechaHora.getMinutes();
var segundos = fechaHora.getSeconds();

var sufijo = ' am';
if(horas > 12) {
  horas = horas - 12;
  sufijo = ' pm';
}

if(horas < 10) { horas = '0' + horas; }
if(minutos < 10) { minutos = '0' + minutos; }
if(segundos < 10) { segundos = '0' + segundos; }

document.getElementById("reloj").innerHTML =
horas+':'+minutos+':'+segundos+sufijo;

<div id="reloj" />
```


Utilizando el código anterior, el reloj ya no muestra la hora como `20:14:03`, sino que la muestra en formato de 12 horas y con el sufijo adecuado:

```
08:14:03 pm
```

Para completar el reloj, sólo falta que se actualice su valor cada segundo. Para conseguirlo, se deben utilizar unas funciones especiales de JavaScript que permiten ejecutar determinadas instrucciones cuando ha transcurrido un determinado espacio de tiempo.

La función `setTimeout()` permite ejecutar una función una vez que haya transcurrido un periodo de tiempo indicado. La definición de la función es:

```
setTimeout(nombreFuncion, milisegundos);
```

La función que se va a ejecutar se debe indicar mediante su nombre sin paréntesis y el tiempo que debe transcurrir hasta que se ejecute se indica en milisegundos. De esta forma, si se crea una función encargada de mostrar la hora del reloj y se denomina `muestraReloj()`, se puede indicar que se ejecute dentro de 1 segundo mediante el siguiente código:

```
function muestraReloj() {
  var fechaHora = new Date();
  var horas = fechaHora.getHours();
  var minutos = fechaHora.getMinutes();
  var segundos = fechaHora.getSeconds();

  if(horas < 10) { horas = '0' + horas; }
  if(minutos < 10) { minutos = '0' + minutos; }
  if(segundos < 10) { segundos = '0' + segundos; }

  document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;
}

setTimeout(muestraReloj, 1000);

<div id="reloj" />
```


No obstante, el código anterior simplemente muestra el contenido del reloj 1 segundo después de que se cargue la página, por lo que no es muy útil. Para ejecutar una función de forma periódica, se utiliza una función de JavaScript muy similar a `setTimeout()` que se denomina `setInterval()`. Su definición es:

```
setInterval(nombreFuncion, milisegundos);
```


La definición de esta función es idéntica a la función `setTimeout()`, salvo que en este caso, la función programada se ejecuta infinitas veces de forma periódica con un lapso de tiempo entre ejecuciones de tantos milisegundos como se hayan establecido.

Así, para construir el reloj completo, se establece una ejecución periódica de la función `muestraReloj()` cada segundo:

```
function muestraReloj() {
  var fechaHora = new Date();
  var horas = fechaHora.getHours();
  var minutos = fechaHora.getMinutes();
  var segundos = fechaHora.getSeconds();

  if(horas < 10) { horas = '0' + horas; }
  if(minutos < 10) { minutos = '0' + minutos; }
  if(segundos < 10) { segundos = '0' + segundos; }

  document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;
```

```
}

window.onload = function() {
  setInterval(muestraReloj, 1000);
}

<div id="reloj" />
```

Empleando el objeto `Date` y sus funciones, es posible construir *"cuentras atrás"*, es decir, relojes que muestran el tiempo que falta hasta que se produzca un evento. Además, las funciones `setTimeout()` y `setInterval()` pueden resultar muy útiles en otras técnicas de programación.

ANEXO VI - Calendario

Cuando una aplicación muestra un formulario para que el usuario inserte sus datos, la información más complicada de obtener siempre suelen ser los números de teléfono y las fechas. El motivo es que existen muchas formas diferentes de indicar estos datos. Las fechas por ejemplo se pueden indicar como `dd/mm/aa`, `dd/mm/aaaa`, `aaaa/mm/dd`, `dd-mm-aa`, `dd mm aaaa`, etc.

Los métodos más utilizados para que el usuario introduzca la información relativa a una fecha suelen ser un cuadro de texto en el que tiene que insertar la fecha completa (indicándole el formato que debe seguir) o un cuadro de texto para el día, una lista desplegable para el mes y otro cuadro de texto para el año.

En cualquier caso, para el usuario suele ser más cómodo que la aplicación incluya un calendario en el que pueda indicar la fecha pinchando sobre el día elegido:

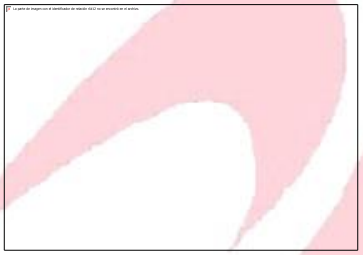


Figura Aspecto de un calendario creado con JavaScript

Además, este método es menos propenso a cometer errores, ya que si el calendario está bien construido, no es posible introducir fechas inválidas y tampoco es posible insertar las fechas en un formato incorrecto.

No obstante, realizar un calendario completo en JavaScript no es una tarea trivial, por lo que no se va a estudiar su desarrollo completo. Afortunadamente, existen scripts gratuitos para mostrar calendarios y que permiten su uso libre incluso en aplicaciones comerciales.

De entre todos los calendarios disponibles, uno de los más completos es el desarrollado por la empresa DynArch, que se puede acceder desde su página web oficial: http://www.dynarch.com/projects/calendar/ y que se puede descargar gratuitamente desde http://sourceforge.net/projects/jscalendar/ El archivo descargado incluye todos los scripts necesarios, su documentación, ejemplos de uso, diferentes estilos CSS para el calendario, etc.

A continuación se indican los pasos necesarios para incluir un calendario básico en cualquier página web:

1) Enlazar los archivos JavaScript y CSS requeridos:

Se descomprime el archivo descargado, se guardan los archivos JavaScript y CSS en el sitio adecuado (en este ejemplo se supone que los archivos JavaScript se guardan en el directorio `js/` y los archivos CSS en el directorio `css/`) y se enlazan directamente desde la cabecera de la página HTML.

```
<head>
...
<style type="text/css">@import url("css/calendar-estilo.css");</style>
<script type="text/javascript" src="js/calendar.js" />
<script type="text/javascript" src="js/calendar-es.js" />
<script type="text/javascript" src="js/calendar-setup.js" />
...
</head>
```

El calendario incluye traducciones a más de 40 idiomas, entre los que se encuentra el español. Para mostrar el calendario en un determinado idioma, tan solo es necesario enlazar el archivo del idioma correspondiente. Las traducciones se encuentran en el directorio `lang` y su formato es `calendar-XX.js`, donde `XX` es el código del idioma.

2) Añadir el código XHTML necesario para el elemento que va a mostrar el calendario:

```
<p>Introduce la fecha pulsando sobre la imagen del calendario</p>
<input type="text" name="date" id="fecha" readonly="readonly" />
<img src="calendario.png" id="selector" />
```

En este caso, el calendario está formado por dos elementos:

- Un cuadro de texto llamado `fecha` y que almacena el valor introducido por el usuario. Como se le ha asignado un atributo `readonly="readonly"`, el usuario no puede modificar su valor.
- Una pequeña imagen o icono que se utiliza para activar el calendario. Cuando el usuario pincha con el ratón sobre la imagen, se muestra el calendario de JavaScript.

3) Configurar el calendario:

```
<script type="text/javascript">
window.onload = function() {
  Calendar.setup({
    inputField: "fecha",
    ifFormat:   "%d / %m / %Y",
    button:     "selector"
  });
}
</script>
```

Una vez enlazados los archivos del calendario y preparado el código XHTML, es necesario inicializar y configurar el calendario. La configuración del calendario consiste en establecer el valor de alguna de sus propiedades. Las propiedades básicas imprescindibles son:

- `inputField`: se trata del atributo `id` del elemento en el que se mostrará la fecha seleccionada, en este ejemplo sería `fecha`.

- `ifFormat`: formato en el que se mostrará la fecha una vez seleccionada (en este caso se ha optado por el formato `dd / mm / aaaa`).
- `button`: atributo `id` del elemento que se pulsa (botón, imagen, enlace) para mostrar el calendario de selección de fecha. En este ejemplo, el `id` de la imagen es `selector`.

Después de esta configuración básica, el calendario ya puede utilizarse en la aplicación:

1) Aspecto por defecto del selector de calendario:

Figura Elementos XHTML del calendario JavaScript: cuadro de texto e icono

2) Al pinchar una vez sobre la imagen del calendario:

Figura Aspecto inicial del calendario JavaScript cuando se abre por primera vez

Se muestra el calendario con el aspecto e idioma establecidos y aparece resaltada la fecha del día actual.

3) Se selecciona la fecha deseada:

Figura Seleccionando una fecha en el calendario JavaScript

4) Después de pinchar con el ratón sobre la fecha deseada:

Figura El cuadro de texto muestra la fecha establecida por el usuario con el calendario JavaScript

Al pulsar con el ratón sobre la fecha deseada, el calendario se cierra automáticamente y el cuadro de texto muestra la fecha seleccionada con el formato indicado en la configuración del calendario. Si se vuelve a abrir el calendario para seleccionar otra fecha, se mostrará resaltada la fecha del día seleccionado y no la fecha del día actual.

ANEXO VII - Tooltip

Los *tooltips* son pequeños recuadros de información que aparecen al posicionar el ratón sobre un elemento. Normalmente se utilizan para ofrecer información adicional sobre el elemento seleccionado o para mostrar al usuario pequeños mensajes de ayuda. Los *tooltips* son habituales en varios elementos de los sistemas operativos:

Figura Aspecto de un tooltip típico en el Sistema Operativo

Realizar un *tooltip* completo mediante JavaScript es una tarea muy compleja, sobre todo por la dificultad de mostrar el mensaje correctamente en función de la posición del ratón. Afortunadamente, existen scripts que ya están preparados para generar cualquier tipo de *tooltip*. La librería que se va a utilizar se denomina overLIB, y se puede descargar desde su página web principal: http://www.bosrup.com/web/overlib/

La descarga incluye todos los scripts necesarios para el funcionamiento del sistema de *tooltips*, pero no su documentación, que se puede consultar en: http://www.bosrup.com/web/overlib/?Documentation. La referencia de todos los comandos disponibles se puede consultar en: http://www.bosrup.com/web/overlib/?Command_Reference

A continuación se indican los pasos necesarios para incluir un *tooltip* básico en cualquier página web:

1) Enlazar los archivos JavaScript requeridos:

```
<script type="text/javascript" src="js/overlib.js"><!-- overLIB (c) Erik
Bosrup --></script>
```

Se descomprime el archivo descargado, se guarda el archivo JavaScript en el sitio adecuado (en este ejemplo se supone que los archivos JavaScript se guardan en el directorio `js/`) y se enlaza directamente desde la cabecera de la página HTML. Los *tooltips* sólo requieren que se enlace un único archivo JavaScript (`overlib.js`). El comentario que incluye el código XHTML es el aviso de copyright de la librería, que es obligatorio incluirlo para poder usarla.

2) Definir el texto que activa el *tooltip* y su contenido:

```
<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba
de un tooltip básico y muy sencillo.');" onmouseout="return
nd();">consectetuer adipiscing elit</a>. Etiam eget metus. Proin varius auctor
tortor. Cras augue neque, porta vitae, vestibulum nec, pulvinar id, nibh.
Fusce in arcu. Duis vehicula nonummy orci.</p>
```

Los tooltip se incluyen como enlaces de tipo `<a>` para los que se definen los eventos `onmouseover` y `onmouseout`. Cuando el ratón se pasa por encima del enlace anterior, se muestra el *tooltip* con el aspecto por defecto:

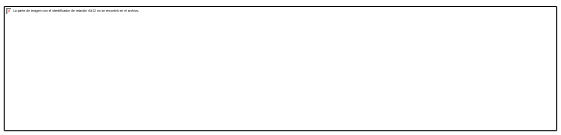


Figura Aspecto por defecto del tooltip creado con la librería overLIB

La librería overLIB permite configurar completamente el aspecto y comportamiento de cada *tooltip*. Las opciones se indican en cada *tooltip* y se incluyen como parámetros al final de la llamada a la función `overlib()`:

```
<a href="#" onmouseover="return overlib('Prueba de un tooltip básico y muy
sencillo.', CENTER);" onmouseout="return nd();">consectetuer adipiscing
elit</a>
```

El parámetro `CENTER` indica que el *tooltip* se debe mostrar centrado respecto de la posición del ratón:

Figura Centrando el tooltip respecto de la posición del ratón

Otra opción que se puede controlar es la anchura del *tooltip*. Por defecto, su anchura es de `200px`, pero la opción `WIDTH` permite modificarla:

```
<a href="#" onmouseover="return overlib('Prueba de un tooltip básico y muy
sencillo.', CENTER, WIDTH, 120);" onmouseout="return nd();">consectetuer
adipiscing elit</a>
```

El valor de la nueva anchura se indica en el parámetro que va detrás de `WIDTH`. El nuevo aspecto del *tooltip* es el siguiente:

Figura Definiendo la anchura que muestra el tooltip

El uso de los parámetros de configuración en la propia llamada a la función que muestra los *tooltips* no es recomendable cuando el número de parámetros empieza a ser muy numeroso. Afortunadamente, overLIB permite realizar una única configuración que se utilizará para todos los *tooltips* de la página.

La configuración global consiste en llamar a la función `overlib_pagedefaults()` con todos los parámetros de configuración deseados. Por tanto, el código JavaScript necesario para realizar los cambios configurados hasta el momento sería:

```
<script type="text/javascript" src="js/overlib.js"><!-- overLIB (c) Erik
Bosrup --></script>
<script type="text/javascript">
overlib_pagedefaults(CENTER, WIDTH, 120);
</script>

<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba
de un tooltip básico y muy sencillo.');" onmouseout="return
nd();">consectetuer adipiscing elit</a>. Etiam eget metus. Proin varius auctor
tortor. Cras augue neque, porta vitae, vestibulum nec, pulvinar id, nibh.
Fusce in arcu. Duis vehicula nonummy orci.</p>
```

Utilizando esta configuración global, se va a modificar por último el aspecto del *tooltip* para hacerlo más parecido a los tooltips de los sistemas operativos:

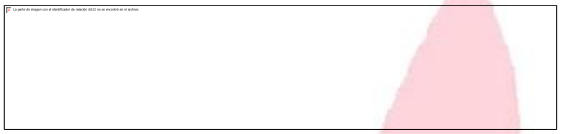


Figura Tooltip creado con la librería overLIB y personalizado para que parezca un tooltip de Sistema Operativo

En el ejemplo anterior se ha modificado el tamaño y tipo de letra y el color de fondo del *tooltip* mediante la siguiente configuración:

```
overlib_pagedefaults(WIDTH,150,FGCOLOR,'#ffffcc',BGCOLOR,'#666666',TEXTFONT,"A
rial, Helvetica, Verdana",TEXTSIZE,".8em");
```