Motor de renderizado

Un motor de renderizado es software que toma contenido marcado (como HTML, XML, archivos de imágenes, etc.) e información de formateo (como CSS, XSL, etc.) y luego muestra el contenido ya formateado en la pantalla de aplicaciones. El motor "pinta" en el área de contenido de una ventana, la cual es mostrada en un monitor o una impresora. Los motores de renderizado se usan típicamente en navegadores web, clientes de correo electrónico, u otras aplicaciones que deban mostrar (y editar) contenidos web.

Todos los navegadores web incluyen necesariamente algún tipo de motor de renderizado. Sin embargo, el término "motor de renderizado" solo alcanzó un uso popular cuando el proyecto Mozilla diseñó el motor de su navegador (Gecko) como un componente aparte del propio navegador. En otras palabras, el motor de Mozilla era reutilizable por otros navegadores diferentes, y mucha gente se empezó a referir a Gecko como un "motor de renderizado" en sí, en lugar de como una parte del navegador.

Algunos de los motores de renderizado más notables son:

- Gecko, utilizado en Mozilla Suite, y otros navegadores como Galeon.
- Trident, el motor de Internet Explorer para Windows.
- KHTML/WebCore, el motor de Konqueror. Antecesor del WebKit.
- Presto, el antiguo motor de Opera.
- Tasman, el motor de Internet Explorer para Mac.
- WebKit, el motor de Firefox, Epiphany y Safari.
- Blink, el nuevo motor de Google Chrome y y Opera (se trata de un fork de WebKit).
- Servo, nuevo motor en desarrollo por parte de Mozilla (con el apoyo de Samsung), está siendo optimizado para la arquitectura ARM y la plataforma Android.

¿Qué es el DOM?

El modelo de objeto de documento (DOM) es una interfaz de programación para los documentos HTML y XML. Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, al fin de modificar, tanto su

estructura, estilo y contenido. El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. Esencialmente, conecta las páginas web a scripts o lenguajes de programación.

Una página web es un documento. Éste documento puede exhibirse en la ventana de un navegador o también como código fuente HTML. Pero, en los dos casos, es el mismo documento. El modelo de objeto de documento (DOM) proporciona otras formas de presentar, guarda y manipular este mismo documento. El DOM es una representación completamente orientada al objeto de la página web y puede ser modificado con un lenguaje de script como JavaScript.

El W3C DOM estándar forma la base del funcionamiento del DOM en muchos navegadores modernos. Varios navegadores ofrecen extensiones más allá del estándar W3C, hay que ir con extremo cuidado al utilizarlas en la web, ya que los documentos pueden ser consultados por navegadores que tienen DOMs diferentes.

Por ejemplo, el DOM de W3C especifica que el método getElementsByTagName en el código de abajo debe devolver una lista de todos los elementos <P> del documento:

```
paragraphs = document.getElementsByTagName("P");
// paragraphs[0] es el primer elemento <p>
// paragraphs[1] es el segundo elemento <p>, etc.
alert(paragraphs[0].nodeName);
```

Todas las propiedades, métodos y eventos disponibles para la manipulación y la creación de páginas web está organizado dentro de objetos. Un ejemplo: el objeto document representa al documento mismo, el objeto table hace funcionar la interfaz especial HTMLTableElement del DOM para acceder a tablas HTML, y así sucesivamente.

DOM y JavaScript

El ejemplo corto de abajo, como casi todos los ejemplos de esta referencia, es JavaScript. Es decir, es escrito en JavaScript pero utiliza el DOM para acceder al documento y a sus elementos. El DOM no es un lenguaje de programación pero sin él, el lenguaje JavaScript no tiene ningún modelo o noción de las páginas web, de la páginas XML ni de los elementos con los cuales es usualmente relacionado. Cada elemento -"el documento íntegro, el título, las tablas dentro del documento, los títulos de las tablas, el texto dentro de las celdas de las tablas"- es parte del modelo de objeto del documento para cada documento, así se puede acceder y manipularlos utilizando el DOM y un lenguaje de escritura, como JavaScript.

En el comienzo, JavaScript y el DOM estaban herméticamente enlazados, pero después se desarrollaron como entidades separadas. El contenido de la página es almacenado en DOM y el acceso y la manipulación se hace vía JavaScript, podría representarse aproximadamente así:

API(web o página XML) = DOM + JS(lenguaje de script)

El DOM fue diseñado para ser independiente de cualquier lenguaje de programación particular, hace que la presentación estructural del documento sea disponible desde un simple y consistente API. Aunque en este manual nos centramos exclusivamente en JavaScript, la directrices del DOM pueden construirse para cualquier lenguaje, así lo demuestra el siguiente ejemplo de Python:

```
# ejemplo DOM de python
import xml.dom.minidom as m
doc = m.parse("C:\\Projects\\Py\\chap1.xml");
doc.nodeName # DOM property of document object;
p_list = doc.getElementsByTagName("para");
```

¿Cómo se accede al DOM?

No se tiene que hacer nada especial para empezar a utilizar el DOM. Los diferentes navegadores tienen directrices DOM distintas, y éstas directrices tienen diversos grados de

conformidad al actual estándar DOM (un tema que se intenta evitar en este manual), pero todos los navegadores web usan el modelo de objeto de documento para hacer accesibles las páginas web al script.

Cuando se crea un script –esté en un elemento <SCRIPT> o incluido en una página web por la instrucción de cargar un script– inmediatamente está disponible para usarlo con el API, accediendo así a los elementos documento o ventana, para manipular el documento mismo o sus diferentes partes, las cuales son los varios elementos de una página web. La programación DOM hace algo tan simple como lo siguiente, lo cual abre un mensaje de alerta usando la función alerta() desde el objeto ventana, o permite métodos DOM más sofisticados para crear realmente un nuevo contenido, como en el largo ejemplo de más abajo.

```
<body onload="window.alert('Bienvenido a mi página!');">
```

Aparte del elemento <script> en el cual JavaScript es definido, el ejemplo siguiente muestra la función a ejecutar cuando el documento se está cargando (y que el DOM completo es disponible para su uso). Esta función crea un nuevo elemento H1, le pone texto y después lo agrega al árbol del documento:

```
<html>
<head>
<script>
// ejecuta esta función cuando la página se carga
window.onload = function() {
   // crea un par de elementos
   // en otra página HTML vacía
   heading = document.createElement("h1");
   heading_text = document.createTextNode("Cabeza grande!");
   heading.appendChild(heading_text);
   document.body.appendChild(heading);
}
</script>
</head>
<body>
</body>
</html>
```

Tipos de datos importantes

Esta parte intenta describir, de la manera más simple posible, los diferentes objetos y tipos. Pero hay que conocer una cantidad de tipos de datos diferentes que son utilizados por el API. Para simplificarlo, los ejemplos de sintaxis en esta API se refieren a nodos como elements, a una lista de nodos (o simples elementos) y a nodos de atributo como attributes.
La siguiente tabla describe brevemente estos tipos de datos.

`document`	Cuando un miembro devuelve un objeto del tipo `document` (por ejemplo, la propiedad **ownerDocument** de un elemento devuelve el documento "`document`" al cual pertenece), este objeto es la raíz del objeto documento en sí mismo.
`element`	`element` se refiere a un elemento o a un nodo de tipo de elemento "`element`" devuelto por un miembro del API de DOM. Dicho de otra manera, por ejemplo, el método `document.createElement()` devuelve un objeto referido a un `nodo`, lo que significa que este método devuelve el elemento que acaba de ser creado en el DOM. Los objetos `element` ponen en funcionamiento a la interfaz `Element` del DOM y también a la interfaz de nodo "`Node`" más básica.
`nodeList`	Una "`nodeList`" es una serie de elementos, parecido a lo que devuelve el método `document.getElementsByTagName()`. Se accede a los items de la `nodeList` de cualquiera de las siguientes dos formas: list.item(1) list[1] Ambas maneras son equivalentes. En la primera, **item()** es el método del objeto `nodeList`. En la última se utiliza la típica sintaxis de acceso a listas para llegar al segundo ítem de la lista.
`attribute`	Cuando un atributo ("`attribute`") es devuelto por un miembro (por ej., por el método **createAttribute()**), es una referencia a un objeto que expone una interfaz particular (aunque limitada) a los atributos. Los atributos son nodos en el DOM igual que los elementos, pero no suelen usarse así.
`NamedNodeMap`	Un `namedNodeMap` es una serie, pero los ítems son accesibles tanto por el nombre o por un índice, este último caso es meramente una conveniencia para enumerar ya que no están en ningún orden en particular en la lista. Un `NamedNodeMap` es un método de ítem() por esa razón, y permite poner o quitar ítems en un `NamedNodeMap`

Interfaces del DOM

Uno de los propósitos de esta guía es minimizar el hablar de interfaces abstractas, heredadas y otros detalles de funcionamiento. Más bien, concentrarse sobre los objetos en el DOM y sobre las actuales cosas que se pueden usar para manipular la jerarquía de DOM. Desde el punto de vista del programador web, es bastante indiferente saber que la representación del objeto del elemento HTML FORM toma la propidedad name desde la interfaz HTMLFormElement pero que las propiedades className se toman desde la propia interfaz HTMLElement. En ambos casos, la propiedad está sólo en el objeto formulario.

Pero puede resultar confuso el funcionamiento de la fuerte relación entre objetos e interfaces en el DOM, por eso esta sección intentará hablar un poquito sobre las interfaces actuales en la especificación del DOM y de cómo se dispone de ellas.

Interfaces y objetos

En algunos casos un objeto pone en ejecución a una sola interfaz. Pero a menudo un objeto toma prestada una tabla HTML (table) desde muchas interfaces diversas. El objeto tabla, por ejemplo, pone en funcionamiento una Interfaz especial del elemento tabla HTML, la cual incluye métodos como createCaption y insertRow. Pero como también es un elemento HTML, table pone en marcha a la interfaz del Element descrita en el capítulo La referencia al elemento del DOM. Y finalmente, puesto que un elemento HTML es también, por lo que concierna al DOM, un nudo en el árbol de nudos que hace el modelo de objeto para una página web o XML, el elemento de tabla hace funcionar la interfaz más básica de Node, desde el cual deriva Element.

La referencia a un objeto table, como en el ejemplo siguiente, utiliza estas interfaces intercambiables sobre el objeto.

```
var table = document.getElementById("table");
var tableAttrs = table.attributes; // Node/Element interface
for(var i = 0; i < tableAttrs.length; i++){
  // Interfaz del elemento de tabla HTML: atributo del borde
```

```
  if(tableAttrs[i].nodeName.toLowerCase() == "border")
    table.border = "1";
}
// Interfaz del elemento de tabla HTML: atributo del sumario
table.summary = "note: increased border";
```

Interfaces esenciales en el DOM

Esta sección lista las interfaces más comúnmente utilizadas en el DOM. La idea no es describir qué hacen estas APIs pero sí dar una idea de las clases de métodos y propiedades que se encuentran con el uso del DOM.

document y window son objetos cuya interfaces son generalmente muy usadas en la programación de DOM. En término simple, el objeto window representa algo como podría ser el navegador, y el objeto document es la raíz del documento en sí. Element hereda de la interfaz genérica Node, y juntos, estas dos interfaces proporcionan muchos métodos y propiedades utilizables sobre los elementos individuales. Éstos elementos pueden igualmente tener interfaces específicas según el tipo de datos representados, como en el ejemplo anterior del objeto table. Lo siguiente es una breve lista de los APIS comunes en la web y en las páginas escritas en XML utilizando el DOM.

- document.getElementById(id)
- element.getElementsByTagName(name)
- document.createElement(name)
- parentNode.appendChild(node)
- element.innerHTML
- element.style.left
- element.setAttribute
- element.element.getAttribute
- element.addEventListener
- window.content
- window.onload
- window.dump

- window.scrollTo

Probando el API del DOM

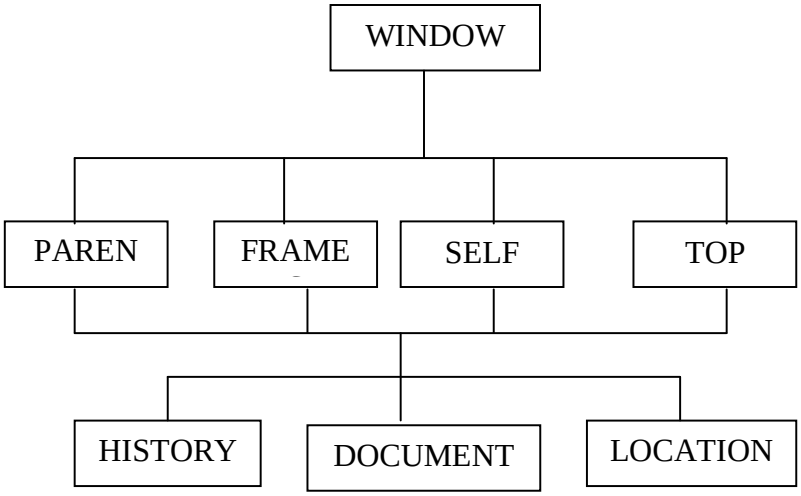
```
<html>
  <head>
    <title>DOM Tests</title>
    <script type="application/javascript">
    function setBodyAttr(attr,value){
    if (document.body) eval('document.body.'+attr+'="'+value+'"');
    else notSupported();
    }
    </script>
  </head>
  <body>
    <div style="margin: .5in; height: 400;">
      <p><b><tt>text</tt>color</b></p>
      <form>
        <select onChange="setBodyAttr('text',
        this.options[this.selectedIndex].value);">
          <option value="black">black
          <option value="darkblue">darkblue
        </select>
        <p><b><tt>bgColor</tt></b></p>
        <select onChange="setBodyAttr('bgColor',
        this.options[this.selectedIndex].value);">
          <option value="white">white
          <option value="lightgrey">gray
        </select>
        <p><b><tt>link</tt></b></p>
        <select onChange="setBodyAttr('link',
        this.options[this.selectedIndex].value);">
          <option value="blue">blue
          <option value="green">green
        </select>  <small>
        <a href="http://www.brownhen.com/dom_api_top.html" id="sample">
        (sample link)</a></small><br>
      </form>
      <form>
        <input type="button" value="version" onclick="ver()" />
      </form>
    </div>
  </body>
</html>
```

LOS OBJETOS DEL NAVEGADOR

Podemos dividir los objetos del navegador en dos grupos: los objetos del nivel más alto en la jerarquía de JavaScript, que tratan directamente con el navegador, con sus particularidades y posibilidades de funcionamiento (como son el abrir una nueva ventana, ver la lista de sitios por donde hemos pasado, obtener datos sobre el propio navegador, la URL en la que estamos trabajando...) y los objetos que tienen una correlación directa con las etiquetas HTML, y que tienen, relación con la programación y creación de las páginas HTML.

Los navegadores proporcionan objetos distribuidos a través de una jerarquía en la que en la parte superior aparece el objeto window.

Esta jerarquía es una forma de dividir los objetos que usted puede utilizar a través de un lenguaje de guiones como JavaScript. Cada uno de estos objetos tiene sus propias propiedades, métodos y eventos.



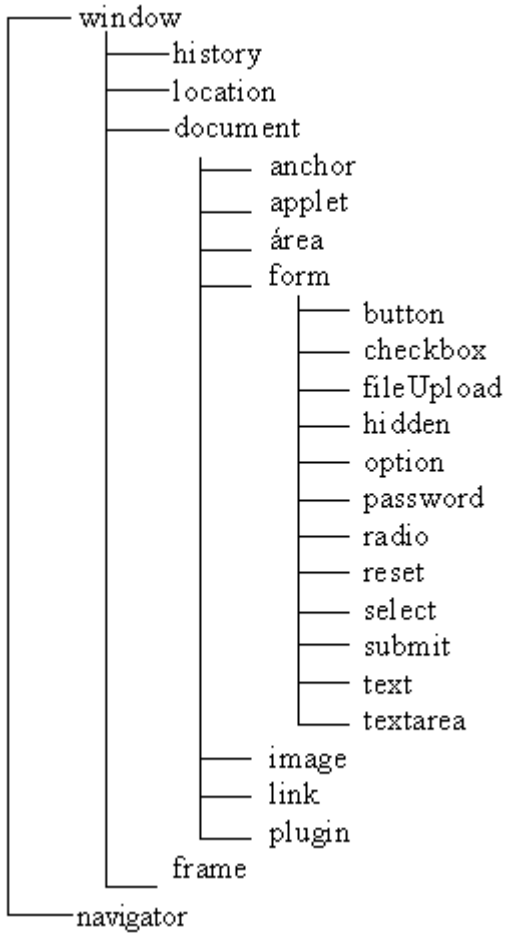
Sin embargo, a diferencia de otros lenguajes orientados a objetos, los objetos del navegador son una jerarquía de composición. Esto quiere decir que los objetos situados en un nivel inferior son propiedades de los objetos situados en un nivel superior.

La siguiente tabla muestra los objetos del "navegador" del modelo JavaScript y sus correspondientes etiquetas HTML:

Objeto de JavaScript	**Etiqueta de HTML**
window	
frame	<FRAME>
document	<BODY>
form	<FORM>
button	<INPUT TYPE="button">
checkbox	<INPUT TYPE="checkbox">
hidden	<INPUT TYPE="hidden">
fileUpload	<INPUT TYPE="file">
password	<INPUT TYPE="password">
radio	<INPUT TYPE="radio">
reset	<INPUT TYPE="reset">
select	<SELECT>
submit	<INPUT TYPE="submit">
text	<INPUT TYPE="text">
textarea	<TEXTAREA>
link	<A HREF=" ">
anchor	<A NAME=" ">
applet	<APPLET>
image	<IMG>
plugin	<EMBED>
área	<MAP>
history	
location	
navigator	

Esta misma tabla se puede contemplar desde un punto de vista jerárquico, atendiendo a la relación contenedor-contenido que presentan los objetos del navegador, tal y como se indicaba en la sección anterior.

El siguiente esquema jerárquico muestra la relación básica de los objetos, clarificando los diferentes elementos y sus situaciones relativas al resto de los objetos:



Es muy importante entender estas relaciones de contenedor-contenido, no solamente en términos de cómo se relacionan los objetos, sino desde un punto de vista práctico, para saber cómo referenciar un objeto. La notación de punto (.) que vimos en este capítulo para hacer referencia a las propiedades y métodos de un objeto, también se utiliza para denotar los objetos que están contenidos dentro de un objeto dado.

Para referenciar una caja de texto dentro de esta estructura deberemos incluir en la referencia los elementos de la jerarquía que la contienen, del siguiente modo:

ventana.documento.formulario.caja_de_texto

Donde ventana es el nombre del objeto window (el nombre por defecto para este objeto es window), documento es el nombre del objeto document (el nombre por defecto para este objeto es document), formulario es el nombre del objeto form y caja_de_texto es

el nombre del objeto text. En la mayoría de los casos se puede ignorar la referencia a la ventana actual (window), pero es necesaria cuando estamos trabajando con múltiples ventanas o trames. Es importante comprender cuando estamos trabajando con un objeto en qué momentos es necesario hacer referencia al objeto contenedor del mismo. Las referencias a todos los objetos que están dentro del objeto document deben incluir a dicho objeto, sino JavaScript no entendería a qué objeto estamos haciendo referencia, aunque ese objeto tuviese un atributo NAME único.

Además de la notación de punto, también podemos utilizar la notación en array. Ésta se utiliza principalmente cuando los objetos a los que queremos hacer referencia no tienen definido su atributo ñame, es decir, no podemos utilizar la notación de punto porque no sabemos cómo identificar al objeto. La forma de utilizar la notación de array para referenciar el primer elemento del primero de los formularios definidos en nuestro documento sería:

document. forms [0] .elements [0]

Vamos a describir someramente los objetos del "navegador", definiendo sus atributos y métodos más básicos.

1.El objeto Window

Es el objeto más alto de la jerarquía de objetos de JavaScript. Todos los elementos con los que el usuario interactúa se sitúan dentro de una ventana. A pesar de que otros objetos puedan no estar presentes, el objeto window siempre existirá.

De hecho, pueden existir múltiples instancias del objeto window en un mismo instante. Por ello, a la hora de manejar varios objetos window desde el código (en entornos con múltiples ventanas o con varios frames) es muy importante saber en todo momento en cuál de ellas estamos trabajando. Cuando se trabaja con entornos de un solo frame se puede ignorar explícitamente la referencia al objeto window, y JavaScript asume que la referencia se da a la ventana actual.

Los objetos de tipo window tienen las siguientes **propiedades**:

- **closed:** Es un valor booleano que indica si la ventana está o no cerrrada.

- **defaultStatus:** Esta propiedad contiene el mensaje por defecto que aparece en la barra de estado del navegador.
- **frames:** Es un array que representa los objetos frame que se encuentran en el objeto window actual. El orden en el que aparecen en el array es el orden en que se definen en el código HTML.
- **history:** Es un array que representa las URL almacenadas en el historial del objeto window actual.
- **length:** Es un valor entero que indica el número de frames del objeto window actual.
- **name:** Contiene el nombre del objeto window actual, o del frame actual.
- **opener:** Contiene la referencia al objeto window que abrió el objeto window actual (si se utilizó para ello el método open()).
- **parent:** Contiene la referencia al objeto window que contiene el frameset.
- **self:** Nombre alternativo del objeto window actual.
- **status:** Determina el mensaje que aparece en la barra de estado del navegador.
- **top:** Nombre alternativo para la ventana de nivel más superior.
- **window:** Nombre alternativo del objeto window actual.

Los objetos de tipo window tienen los siguientes **métodos**:

- **alert(mensaje):** Muestra el mensaje en una caja de diálogo.
- **blur()** Elimina el foco del objeto window actual.
- **clearInterval(id):** Elimina el intervalo referenciado por id.
- **clearTimeout(nombre)** Cancela el intervalo especificado por nombre.
- **Close()** Cierra el objeto window actual.
- **confirm(mensaje):** Muestra el mensaje en una caja de diálogo con las opciones de "Aceptar" y "Cancelar".
- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto window.
- **focus():** Captura el foco sobre el objeto window actual.
- **moveBy(x,y):** Mueve el objeto window actual al número de pixels especificados por x e y.

- **moveTo(x,y):** Mueve el objeto window actual a las coordenadas especificadas por x e y.
- **open(url, nombre, características):** Abre la url especificada en la ventana llamada nombre. Si no existe tal ventana, la url se abrirá en una nueva ventana con las características especificadas. Entre las características opcionales que definen una nueva ventana podemos encontrar las siguientes:
  - **toolbar = [yes,no,l,0]:** Indica si la nueva ventana deberá tener barra de herramientas o no.
  - **location = [yes,no,l,0]:** Indica si la nueva ventana deberá tener campo de localización o no.
  - **directories = [yes,no,l,0]:** Indica si la nueva ventana deberá tener botones de dirección o no.
  - **status = [yes,no,l,0]:** Indica si la nueva ventana deberá tener barra de estado o no.
  - **menubar = [yes,no,l,0]:** Indica si la nueva ventana deberá tener barra de menús o no.
  - **scrollbars = [yes,no,l,0]:** Indica si la nueva ventana deberá tener barras de desplazamiento o no.
  - **resizable = [yes,no,l,0**]: Indica si la nueva ventana podrá cambiar de tamaño o no.
  - **width = pixels:** Indica el ancho de la ventana cliente en pixels.
  - **height = pixels:** Indica el alto de la ventana cliente en pixels.
  - **outerWidth = pixels:** Indica el ancho total de la ventana en pixels.
  - **outerHeight = pixels:** Indica el alto total de la ventana en pixels.
  - **left=pixels:** Indica la distancia en pixels desde el lado izquierdo de la pantalla a la que se deberá colocar la ventana.
  - **top = pixels:** Indica la distancia en pixels desde el lado superior de la pantalla a la que se deberá colocar la ventana.
  - **alwaysRaised = [yes,no,l,0]:** Indica si la ventana deberá permanecer siempre en frente de cualquier otra.

- **z-lock = [yes,no,l,0]:** Indica si se debe bloquear o no la ventana en su z-order actual.

- **prompt(mensaje, respuesta):** Muestra el mensaje en una caja de diálogo que contiene una caja de texto que permite al usuario introducir información. En dicha caja aparece inicialmente la respuesta por defecto, si es que ésta existe. La respuesta introducida por el usuario en la caja de texto se devuelve como una cadena de caracteres.
- **resizeBy(x,y):** Ajusta el tamaño del objeto window actual moviendo su esquina inferior derecha, el número de pixels especificados por x e y.
- **resizeTo(ancho,alto):** Ajusta el tamaño del objeto window actual cambiando los valores outerWidth y outerHeígth a los valores especificados en ancho y alto.
- **scroll(x,y):** Desplaza el objeto window actual a las coordenadas especificadas por x e y.
- **scrollBy(x,y):** Desplaza el objeto window actual el número de pixels especificados por x e y.
- **scrollTo(x,y):** Desplaza el objeto window actual a las coordenadas especificadas por x e y.
- **setInterval(expresión, tiempo):** Evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Este método devuelve un valor que puede ser utilizado como identificativo por el método clearInterval().
- **SetTimeout(expresión, tiempo):** Evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Este método devuelve un valor que puede ser utilizado como identificativo.
- **toString():** Devuelve una cadena que representa al objeto window.
- **value0f():** Convierte el objeto a su tipo primitivo (number, boolean, string, o function).

El siguiente ejemplo muestra la apertura y cierre básicos de las ventanas, a través de los métodos open() y close(). En el código se utiliza una función denominada abrirVentana() que se encarga de crear la nueva ventana y escribir un documento en ella.

Cabe destacar la utilización de la variable miVentana que almacena la referencia a la ventana creada, que devuelve la llamada al método open(), esta referencia es esencial para poder más tarde escribir o trabajar sobre la nueva ventana creada. A la hora de cerrar la ventana, se utiliza la propiedad self de la ventana que hace referencia a la ventana por defecto. La utilización del manejador onClick se verá más adelante:

```
<HTML>
<HEAD><TITLE>Objeto Window</TITLE>
<SCRIPT LANGUAGE ="JavaScript">
<!-- se oculta la información de los navegadores antiguos
  function abrirVentana(){
    var miVentana;
miVentana=open("","miVentana","menubar=0,width=250,height=150,
top=100,left=100");
    miVentana.document.write("<HTML>");
    miVentana.document.write("<HEAD>");
    miVentana.document.write("<TITLE>Nuevo objeto
Window</TITLE>");
    miVentana.document.write("</HEAD>");
    miVentana.document.write("<BODY>");
    miVentana.document.write("<CENTER>");
    miVentana.document.write("<H3>Esta nueva ventana llamada '");
    miVentana.document.write(miVentana.name);
    miVentana.document.write(" ' se cerrará al pulsar el botón...<BR>");
    miVentana.document.write("<FORM>");
    miVentana.document.write('<INPUT TYPE=BUTTON
VALUE="Cerrar" onClick="self.close()">');
    miVentana.document.write("</FORM>")
    miVentana.document.write("</CENTER>");
    miVentana.document.write("</BODY>");
    miVentana.document.write("</HTML>");
```

```
 }
// final del comentario -->
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
<H2>Manejando el objeto window...</H2>
<FORM NAME="miFormulario">
<INPUT TYPE=BUTTON NAME="miBoton" VALUE="Púlsame"
onClick="abrirVentana()">
</FORM>
</CENTER>
</BODY></HTML>
```

O bien:

```
for (p in window.location) console.debug ("propiedad:"+p+"--> "+window.location[p]);
```

2.El objeto Location

El objeto location encapsula la **URL** (uniform Resource Locator) de la página actual. Un URL es una forma estándar de referirnos exactamente a un determinado recurso.

Tiene dos propósitos principales: modificar el objeto location para cambiar a una nueva URL y extraer sus componentes específicos para poder trabajar con ellos de forma individual (la estructura básica de una URL contiene el protocolo a utilizar, el nombre del host servidor, el puerto utilizado y el camino hasta el recurso).

El objeto **location** representa un URL completo. Si recuerda la jerarquía de objetos, sabrá que es una propiedad del objeto window.

Esto quiere decir que puede utilizar la sintaxis **window.location** o simplemente **location,** siempre que no haya peligro de contusión (cuando exista más de un objeto window).

El objeto location pertenece a la ventana que contiene el código JavaScript cuando no se utilizan frames. Sin embargo, en una página dividida en frames, cada uno de éstos tiene su propio objeto location.

Veamos el siguiente ejemplo:

```
<script language="JavaScript">
<!—
document.write("<B>propiedad href: </B>"+location.href+"<BR>")
document.write("<B>propiedad hash: </B>"+location.hash+"<BR>")
document.write("<B>propiedad host: </B>"+location.host+"<BR>")
document.write("<B>propiedad hostname: </B>"+location.host+"<BR>")
document.write("<B>propiedad pathname: </B>"+location.pathname+"<BR>")
document.write("<B>propiedad port: </B>"+location.port+"<BR>")
document.write("<B>propiedad protocol: </B>"+location.protocol+"<BR>")
document.write("<B>propiedad search: </B>"+location.search+"<BR>")
//-->
</script>
```

El objeto location presenta ocho propiedades y dos métodos, los cuales no se suelen utilizar demasiado. En este script lo que hacemos es imprimir en la página Web el valor de estas propiedades.

Los objetos de tipo location tienen las siguientes **propiedades**:

- **hash:** Es una cadena que contiene el nombre del enlace, dentro de la URL.
- **host:** Es una cadena que contiene el nombre del servidor y el número de puerto, dentro de la URL.
- **hostname:** Es una cadena que contiene el nombre de domino del servidor (o la dirección IP), dentro de la URL.
- **href:** Es una cadena que contiene la URL completa.
- **pathname:** Es una cadena que contiene el camino al recurso, dentro de la URL.
- **port:** Es una cadena que contiene el número de puerto del servidor, dentro de la URL.

- **protocol:** Es una cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.
- **search:** Es una cadena que contiene la información pasada en una llamada a un script CGI, dentro de la URL.

Los objetos de tipo location tienen los siguientes **métodos**:

- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto location.
- **reload():** Recarga la URL especificada en la propiedad href del objeto location.
- **replace(cadenaURL):** Reemplaza el historial actual mientras carga la URL especificada en cadenaURL.
- **toString():** Devuelve una cadena que representa al objeto location.
- **valueOf():** Convierte el objeto a su tipo primitivo (number, boolean, string, o function).

Ahora que conocemos el objeto location, vamos a utilizarlo en algún caso práctico. Imagine esta situación: su Web ha cambiado de ubicación en Internet, de forma que ahora la dirección que todos conocían ya no es válida.

¿Cuál es la solución para no perder a sus visitantes? Usted puede seguir manteniendo una página en la ubicación antigua y acceder automáticamente a la nueva dirección.

Veamos cómo hacerlo:

```
<HTML>
<HEAD><TITLE>accediendo a la nueva dirección</TITLE></HEAD>
<BODY onload="location.href=http://www.ole.es>>
<H1>Nos hemos trasladado</H1>
<H1>Accediendo a la nueva direccón</H1>
</BODY>
</HTML>
```

Observe cómo se utiliza la propiedad **href** del objeto location para actualizar la página, de forma que se acceda a otro URL.

Es decir, vamos a actualizar la propiedad href en el momento en que se cargue la página Web (manejador onLoad de la etiqueta **<BODY>).** En este caso, accederemos a la página del buscador Ole.

Observe cómo ha sido necesario incluir la cadena entre comillas simples ya que el valor del manejador de eventos **onLoad** es en sí una cadena. (Si se pregunta por qué se escribe onload en lugar de onLoad, recuerde que está escribiendo código HTML y no JavaScript, por lo que en este caso puede escribirlo como quiera. El código JavaScript siempre aparecerá entre las etiquetas <SCRIPT> y </SCRIPT>).

Con el código mostrado, no es necesaria ninguna intervención del usuario para acceder a la nueva ubicación. Esto es muy conveniente ya que debemos evitar todas las molestias posibles a nuestros visitantes para que no caigan en la tentación de ir a otro sitio de la Web.

3. El objeto History

El objeto history se encarga de almacenar la lista de sitios por los que se ha estado "navegando", es decir, guarda la referencia de los lugares visitados. Lo importante a la hora de estudiar este objeto es entender que usted puede moverse a través de esta lista, pero **no puede acceder a las direcciones** que están en la misma. Esto se debe a motivos de seguridad: nadie tiene por qué conocer qué lugares ha visitado.

Se utiliza este objeto para manejar dicha lista, pudiendo realizar movimientos hacia delante y hacia atrás en la misma.

Los objetos de tipo history tienen las siguientes **propiedades**:

- **current:** Es una cadena que contiene la URL completa de la entrada actual en el historial.
- **next:** Es una cadena que contiene la URL completa de la siguiente entrada en el historial.
- **length:** Es un valor entero que contiene el número de ítems del historial.
- **previous:** Es una cadena que contiene la URL completa de la anterior entrada en el historial.

Los objetos de tipo history tienen los siguientes **métodos**:

- **back():** Recarga la URL del documento anterior dentro del historial.
- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto history.
- **forward():** Recarga la URL del documento siguiente dentro del historial.
- **go(posición):** Recarga la URL del documento especificado por posición dentro del historial. El parámetro posición puede ser un entero, en cuyo caso indica la posición relativa del documento dentro del historial; o una cadena de caracteres, en cuyo caso representa toda o parte de una URL presente en el historial.
- **toString():** Devuelve una cadena que representa al objeto history.
- **value0f():** Convierte el objeto a su tipo primitivo (number, boolean, string, o function).

Veamos un ejemplo:

```
<HTML>
<HEAD><TITLE>Título de la página</TITLE></HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" value="Ir Atras" onclick="history.back()">
<INPUT TYPE="button" value="Ir Adelante"
onclick="history.forward()">
</FORM>
</BODY>
</HTML>
```

4. El objeto Navigator

El objeto navígator no tiene una relación real con el resto de los objetos de la jerarquía de JavaScript. Este objeto nos proporciona información relativa al navegador Web que se está utilizando para visualizar los documentos.

Usted ya sabe que existen diferencias en cómo interpretan el código de las páginas Web los distintos navegadores de Internet.

Esta situación produce muchos quebraderos de cabeza a los diseñadores de páginas Web, siendo uno de los principales problemas con los que se enfrentan.

JavaScript permite acceder al objeto navigator, que no pertenece a la jerarquía de los objetos del navegador, sino que es un objeto independiente.

El propósito de este objeto es facilitar información sobre el propio navegador: nombre, versión, etc. Esto facilita la labor de un diseñador de páginas Web ya que puede decidir qué código ejecutar en función del navegador que se esté utilizando.

Veamos un ejemplo:

```
<script language="JavaScript">
function navegador()
{
document.writelen("<B>Propiedad appName:</B>" +navigator.appname)
document.write("<BR>")
document.writelen("<B>Propiedad appVersion:</B>"
+navigator.appVersion)
document.write("<BR>")
document.writelen("<B>Propiedad appCodeName:</B>"
+navigator.appCodeName)
document.write("<BR>")
document.writelen("<B>Propiedad UserAgent:</B>"
+navigator..userAgent)
document.write("<BR>")
}
</SCRIPT>
```

A continuación puede ver el valor de estas propiedades para el navegador Internet Explorer:

Propiedad appName: *Microsoft Internet Explorer*

Propiedad appVersion: *4.0 (compatible; MSEB4.01; Windows 98)*

Propiedad appCodeName: *Mozilla*

Propiedad userAgent: *Mozilla/4.0 (compatible; MSIE4.01; Windows 98)*

Ahora puede ver el valor de estas propiedades para el navegador Netscape Navigator:

Propiedad appName: *Netscape*

Propiedad appVersion: *4 03 [es] (Win95; I)*

Propiedad appCodeName: Mozilla

Propio dad userAgent: Mozilla/4.03 [es] (Win95; I)

Observe cómo ambos navegadores tienen el mismo nombre en código: **Mozilla,** por lo que deberá utilizar otras propiedades para diferenciar uno de otro.

Al conocer qué navegador está utilizando el usuario, usted puede dirigirlo a un lugar u otro, de forma que las páginas Web se adapten a las características especiales de dicho navegador.

Los objetos de tipo navigator tienen las siguientes **propiedades**:

- **appCodeName:** Es una cadena que contiene el nombre en código del navegador.
- **appName:** Es una cadena que contiene el nombre del cliente. Es el nombre normal del navegador. Por ejemplo, Microsoft Internet Explorer o Netscape
- **appVersión: Es una cadena que contiene información sobre la versión del navegador. Por ejemplo, 2.0, 3.0, 4.0, etc. El formato utilizado es el siguiente: númeroVersion(plataforma; país).**
- **mimeTypes:** Es un array que contiene todos los tipos MIME soportados por el cliente.
- **plugins:** Es un array que contiene todos los plug-ins soportados por el cliente.
- **userAgent:** Es una cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades appCodeName y appVersion.

Los objetos de tipo navigator tienen los siguientes **métodos**:

- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto navigator.

- **javaEnabled():** Devuelve true si el cliente permite la utilización de Java; en otro caso, devuelve false.
- **taintEnabled():** Devuelve true si el cliente permite la contaminación de datos; en otro caso, devuelve false.
- **toString():** Devuelve una cadena que representa al objeto navigator.
- **valueOf():** Convierte el objeto a su tipo primitivo (number, boolean, string, o function).

Ojo: Internet explorer:

```
<script type="text/javascript">
if (window.clientInformation.javaEnabled() == true )
    // Java is enabled; applets can run.
</script>
```

El siguiente ejemplo muestra las características básicas del navegador que está siendo utilizado, para ello hace uso de las propiedades del objeto navigator:

```
<HTML>
<HEAD><TITLE>Objeto navigator</TITLE></HEAD>
<BODY>
<CENTER>
<SCRIPT LANGUAGE ="JavaScript">
<!-- se oculta la información de los navegadores antiguos
 document.write("<H2>Manejando el objeto navigator...</H2>");
 document.write("<BR>");
 document.write("<TABLE BORDER=1 CELLPADDING=4>");
 document.write("<TR ALIGN=CENTER><TD> navegador </TD>");
 document.write("<TD>" + navigator.appName + "</TD></TR>");
 document.write("<TR ALIGN=CENTER><TD> código </TD>");
 document.write("<TD>" + navigator.appCodeName +
"</TD></TR>");
 document.write("<TR ALIGN=CENTER><TD> versión </TD>");
 document.write("<TD>" + navigator.appVersion + "</TD></TR>");
```

```
  document.write("<TR ALIGN=CENTER><TD> agente </TD>");
  document.write("<TD>" + navigator.userAgent + "</TD></TR>");
  document.write("</TABLE>");
  document.write("<H2>...Hecho</H2>");
// final del comentario -->
</SCRIPT>
</CENTER>
</BODY>
</HTML>
```

5.El objeto Document

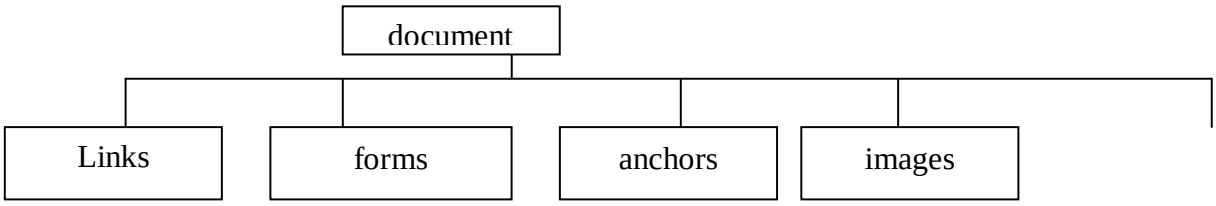
A pesar de que el objeto window es el de nivel más alto en la jerarquía de JavaScript, su función principal es la de ser el contenedor del resto de los objetos, no tiene un contenido propio. El contenido del documento Web que se está visualizando realmente está asociado al objeto document. El objeto document, como ya vimos, es el equivalente a la etiqueta <BODY> de HTML; es, por tanto, esencial en la programación del código que conforma el documento, pues todas las acciones que ocurren dentro de una página Web se dan dentro del objeto document.

El objeto document es el objeto más importante dentro de la jerarquía de objetos del navegador ya que representa el contenido de la página Web.

La mayor parte de la interacción entre el usuario y el script se realiza a través del objeto document, que da acceso a todo el contenido de la página: texto, imágenes, hipervínculos, etc.

No debe pensar que el objeto document representa sólo aquello que está en el cuerpo de la página Web. También incluye los elementos situados en su cabecera. Además, como es una propiedad del objeto window, existe un objeto document por cada objeto window. Esto quiere decir que, si trabaja con frames, tendrá distintos objetos document.

Además, el objeto document posee entre sus propiedades otros objetos. Por ejemplo, si incluye uno o más formularios en la página Web, estará creando objetos **form.** Lo mismo ocurre con las imágenes, hipervínculos o marcadores.



Observe la imagen adjunta, donde se muestra la jerarquía de objetos ampliada. Como puede ver, el objeto **document** está compuesto por varios objetos, situados en un nivel inferior.

En muchos casos, una propiedad del objeto document no se evalúa a un objeto individual sino a un **conjunto de objetos.** Por ejemplo, el objeto document está compuesto por un conjunto de imágenes, un conjunto de hipervínculos, un conjunto de formularios, etc.

Esto quiere decir que, al acceder a una propiedad del objeto document. nos podemos encontrar con que dicha propiedad se evalúe a más de un objeto. En este caso. será necesaria una forma de acceder a un objeto en particular de dicho conjunto. Esta forma será a través de las **matrices,** como ya veremos.

En esta lección aprenderá a acceder a las **propiedades** que pueden cambiar el aspecto estético de la página Web: título, colores, etc. Después veremos cómo crear páginas Web enteras a partir de un script utilizando los **métodos** del objeto document y finalmente veremos cómo acceder al **conjunto de imágenes** del documento.

Los objetos de tipo document tienen las siguientes **propiedades**, que reflejan los atributos de un documento HTML en JavaScript:

- **alinkColor:** Representa el color de los enlaces activos.
- **anchors:** Es un array que contiene todos los enlaces internos que existen dentro del documento.
- **applets:** Es un array que contiene todos los applets que existen dentro del documento.
- **bgColor:** Representa el color de fondo del documento.
- **cookie:** Es una cadena que contiene los valores de las cookies para el documento actual.
- **domain:** Es el nombre del servidor que ha servido el documento.
- **embeds:** Es un array que contiene todos los elementos <EMBED> que existen dentro del documento.
- **fgColor:** Representa el color de primer plano del documento.

- **forms:** Es un array que contiene todos los objetos form que existen dentro del documento.
- **images:** Es un array que contiene todas las imágenes que existen dentro del documento.
- **lastModified:** Es una cadena que contiene la fecha en que se realizó la última modificación sobre el documento.
- **linkColor:** Representa el color de los enlaces del documento.
- **links:** Es un array que contiene todas los enlaces extemos que existen dentro del documento.
- **location:** Es una cadena que contiene la URL del documento actual.
- **referrer:** Es una cadena que contiene la URL del documento que llamó al actual, si el usuario utilizó un enlace.
- **title:** Es una cadena que contiene el título del documento actual.
- **vlinkColor:** Representa el color de los enlaces visitados del documento.

Los objetos de tipo document tienen los siguientes **métodos**:

- **clear():** Limpia la ventana del documento.
- **close():** Cierra la escritura sobre el documento actual.
- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto document.
- **Open(mime,"replace"):** Abre la escritura sobre un documento. El parámetro mime especifica el tipo de documento soportado por el navegador. Si la cadena "replace " se pasa como segundo parámetro, se reutiliza el documento anterior dentro del historial.
- `var win = window.open("", "win", "width=300,height=200"); // a window object`

```
with (win.document) {
  open("text/html", "replace");
  write("<HTML><HEAD><TITLE>New Document</TITLE></HEAD><BODY>Hello,
world!</BODY></HTML>");
  close();
}
```

- **toString():** Devuelve una cadena que representa al objeto document.

- **valueOf():** Convierte el objeto a su tipo primitivo (number, boolean, string, o function).
- **write():** Escribe texto y HTML sobre el documento actual.
- **writeln():** Escribe texto y HTML sobre el documento actual, seguido de una nueva línea.

Veamos el siguiente ejemplo:

```
<HTML>
<HEAD><TITLE>cambiar el color del documento</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!—
function cambiarfondovalor() { document.bgcolor=valor }
function cambiarlink(valor){document.linkcolor=valor }
//-->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
var bg=prompt("Introduzca el valor hexadecimal del color de fondo de la página.","#ffffff")
cambiarfondo(bg)
var li=prompt("Introduzca el valor hexadecimal del color de los hipervínculos.","#000000")
cambiarlink(li)
</SCRIPT>
<A HREF="dos.htm"> Pulse aquí para ir a dos.htm</a>
</BODY>
</HTML>
```

En esta página Web se incluye un hipervínculo a una determinada página Web. El resto del código está dividido en dos scripts.

En la cabecera de la página se definen dos funciones. **CambiarFondo** modifica la propiedad **bgColor** del objeto document. Con esta propiedad establecerá el color del fondo de la página.

La función **CambiarLink** hace lo mismo para la propiedad **linkCoIor,** que se encarga del color de los hipervínculos.

Existen otras propiedades que afectan al color de los elementos. Así, con **bgColor** establecerá el color del texto, **alinkCoIor** el color de los hipervínculos activos y **vlinkColor,** el color de los hipervínculos visitados.

En el cuerpo de la página se sitúa el segundo script. En este caso, lo único que se hace es pedirle al usuario que introduzca el valor hexadecimal correspondiente al color que desea utilizar.

En ocasiones, las propiedades de un objeto no se evalúan a un único valor sino a un **conjunto** de valores u objetos. Este es el caso de la propiedad **images** del objeto document. Esta propiedad representa el conjunto de imágenes que aparecen en la página Web.

En esta situación, es necesaria una forma de acceder a uno de los objetos de dicho conjunto. Esta forma es a través de una **matriz.**

JavaScript guarda la referencia a cada imagen en la propiedad images. Esta propiedad es como un conjunto ordenado en el que cada imagen tiene asignado un determinado número.

Así, la primera imagen que aparece en la página tiene el número **0,** la siguiente tiene el número 1 y así sigue hasta la última imagen.

Realmente, la propiedad images es una **matriz.** En una matriz cada uno de sus elementos posee un número o **índice** que lo identifica. Además, una matriz presenta algunas propiedades adicionales, como la posibilidad de conocer cuál es el número de elementos que contiene en un momento dado.

La forma de acceder a los elementos de una matriz es a través de la sintaxis **nombre[índice].**

Es decir, utilizamos el nombre de la matriz e indicamos, entre corchetes, el índice del elemento deseado.

Por ejemplo, si quisiera acceder a la primera imagen de la página, debería escribir el código document.images[0]. Recuerde que la numeración empieza siempre desde 0.

Veamos el siguiente ejemplo:

```
<HTML>
<HEAD><TITLE>Cambiar las imágenes</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function cambiarimagen(valor,imagen){
        document.images[valor].src=imagen
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Selecione la opción deseada</H1>
<TABLE>
<TR>
<TD WIDTH="42"><IMG SRC="uno.gif" WIDTH="49" HEIGHT="23"
onmouseover="cambiarimagen(0,'dos.gif')"
onmouseout="cambiaimagen(0,'uno.gif')"><TD>
<TD WIDTH="100%"><b>Contrareembolso</B></TD>
</TR>
<TR>
<TD WIDTH="42"><IMG SRC="uno.gif" WIDTH="49" HEIGHT="23"
onmouseover="cambiarimagen(1,'dos.gif')"
onmouseout="cambiaimagen(1,'uno.gif')"><TD>
<TD WIDTH="100%"><b>Pago con tarjeta</B></TD>
</TR>
<TD WIDTH="42">< IMG SRC="uno.gif" WIDTH="49"
HEIGHT="23" onmouseover="cambiarimagen(2,'dos.gif')"
onmouseout="cambiaimagen(2,'uno.gif')"><TD>
<TD WIDTH="100%"><b>Giro Postal</B></TD>
</TR>
<TR>
```

```
<TD width="42"><IMG SRC="uno.gif" WIDTH="49" HEIGHT="23"
onmouseover="cambiarimagen(3,'dos.gif')"
onmouseout="cambiaimagen(3,'uno.gif')"><TD>
<TD WIDTH="100%"><b>Cheque bancario</B></TD>
</TR>
</TABLE></BODY>
</HTML>
```

En la pagina Web del ejemplo aparecen cuatro imágenes, por lo que la forma de acceder a ellas será a través del código **document.images[il,** donde i irá desde 0 hasta 3.

El propósito del script es crear una función que cambie la propiedad **src** de las imágenes al pasar el puntero del ratón por encima de ellas. Recuerde que cada imagen es en si un objeto y, por lo tanto, puede tener propiedades.

Así, asociamos la función **Cambiarlmages** con los manejadores de eventos **onmouseOver** (cuando se pasa el puntero del ratón por encima de la imagen) y **onmouseOut** (cuando el puntero del ratón sale de la imagen).

Cuando ocurren estos eventos, se ejecuta la función que cambia la propiedad **src** de la imagen adecuada.

Por ejemplo, cuando ocurre que se ejecuta el manejador **onmouseOver** de la primera imagen, se esta llamando a la función **Cambiarlmages(0,"dos.gif").** Esta función ejecutara la línea **document.images[0].src="dos.gif",** con lo que cambiara la imagen que se muestra en pantalla.

Lo importante es en tender que las imágenes que usted inserta en la pagina son objetos de JavaScript a los que podrá acceder a través de la propiedad images del objeto document.

Existen otras propiedades del objeto document que tienen la misma naturaleza que images.

Así, la propiedad **anchors** es una matriz que permite acceder a los objetos marcador definidos en la pagina. De la misma forma, **links** permite acceder a los objetos hipervínculo, **forms** a los objetos formulario, etc.

LOS OBJETOS DEL LENGUAJE

Además de estos objetos propios del navegador, JavaScript tiene una serie de objetos propios del lenguaje: String, Array, Boolean, Number, Function, Math y Date. Este segundo conjunto de objetos nunca aparece de forma visual, pero se utilizan a la hora de escribir el código: son construcciones del lenguaje JavaScript.

1 El objeto String()

El objeto String nos ofrece distintas formas de manejar cadenas de texto. Siempre que asignamos cadenas de caracteres a una variable o a una propiedad en realidad estamos creando objetos de tipo String. No es necesario declarar los objetos de tipo String, puesto que en el momento de la asignación se crea el objeto.

Los objetos de tipo String tienen las siguientes **propiedades**:

- **length:** Es un valor numérico que nos indica la longitud en caracteres que tiene la cadena dada.
- **prototype:** Nos permite asignar nuevas propiedades al objeto String.

Los objetos de tipo String tienen los siguientes **métodos**:

- **anchor(nombre):** Crea un enlace asignando al atributo NAME el valor de nombre. <A>
- **big():** Muestra la cadena de caracteres con una fuente grande. <BIG>
- **blink():** Muestra la cadena de caracteres con un efecto intermitente. <BLINK>
- **charAt(índice):** Devuelve el carácter situado en la posición especificada por índice.
- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto String.
- **fíxed():** Muestra la cadena de caracteres con una fuente proporcional. <FIXED>
- **fontColor(color):** Cambia el color con el que se muestra la cadena.
- **fontSize(towawo):** Cambia el tamaño con el que se muestra la cadena.
- **indexOf(cadena_buscada,índice):** Devuelve la posición de la primera ocurrencia de cadenaJbuscada dentro de la cadena actual, a partir de la posición indicada por

índice. El argumento índice es opcional, si no se proporciona, la búsqueda comienza por el primer carácter de la cadena.

- **italics():** Muestra la cadena de caracteres en cursiva. **<I>**
- **lastIndexOf(cadena_buscada,índice):** Devuelve la posición de la última ocurrencia de cadena_buscada dentro de la cadena actual, a partir de la posición indicada por índice y buscando hacia atrás. El argumento índice es opcional, si no se proporciona, la búsqueda comienza por el último carácter de la cadena.
- **link(url):** Convierte la cadena en un vínculo asignando al atributo HREF el valor de url.
- **small():** Muestra la cadena de caracteres con una fuente pequeña. <SMALL>
- **split(separador):** Parte la cadena en un array de caracteres. Si el carácter separador no se encuentra, devuelve un array con un solo elemento que coincide con la cadena original.
- **strike():** Muestra la cadena de caracteres tachada. <STRIKE>
- **sub():** Muestra la cadena de caracteres con formato de subíndice. <SUB>
- **substring(primerIndice,segundoIndice):** Devuelve la cadena equivalente a la subcadena que comienza en la posición indicada por primerindice y que finaliza en la posición anterior a la indicada por segundoindice. En el caso de que aquél sea mayor que éste, la cadena comienza por segundoindice y finaliza por el carácter anterior a prímerindice.
- **sup():** Muestra la cadena de caracteres con formato de superíndice. <SUP>
- **toLowerCase():** Devuelve la cadena de caracteres en minúsculas.
- **toString():** Devuelve una cadena que representa al objeto string.

```
<HTML>
<HEAD><TITLE>String's</TITLE></HEAD>
<BODY>
<CENTER>
<SCRIPT LANGUAGE ="JavaScript">
<!-- se oculta la información de los navegadores antiguos
  var cadena="Esta es la cadena de pruebas"
```

```
  document.write("<H3>Manejando Cadenas</H3>");
  document.write("<TABLE>");
  document.write("<TR><TD>...con formato 'BIG':</TD>");
  document.write("<TD>"+ cadena.big() +"</TD></TR>");
  document.write("<TR><TD>...con formato 'SMALL':</TD>");
  document.write("<TD>"+ cadena.small() +"</TD></TR>");
  document.write("<TR><TD>...con formato 'BLINK':</TD>");
  document.write("<TD>"+ cadena.blink() +"</TD></TR>");
  document.write("<TR><TD>...con formato 'B':</TD>");
  document.write("<TD>"+ cadena.bold() +"</TD></TR>");
  document.write("<TR><TD>...con formato 'I':</TD>");
  document.write("<TD>"+ cadena.italics() +"</TD></TR>");
  document.write("<TR><TD>...con formato 'STRIKE':</TD>");
  document.write("<TD>"+ cadena.strike() +"</TD></TR>");
  document.write("<TR><TD>...con formato 'SUP':</TD>");
  document.write("<TD>"+ cadena.sup() +"</TD></TR>");
  document.write("<TR><TD>...con formato 'SUB':</TD>");
  document.write("<TD>"+ cadena.sub() +"</TD></TR>");
  document.write("</TABLE>");
  document.write("<H3>...Hecho</H3>");
// final del comentario -->
</SCRIPT>
</CENTER>
</BODY>
</HTML>
```

El siguiente ejemplo muestra el manejo de cadenas, a través de un conjunto de métodos básicos:

```
<HTML>
<HEAD><TITLE>String's</TITLE></HEAD>
```

```
<BODY>
<CENTER>
<SCRIPT LANGUAGE ="JavaScript">
<!-- se oculta la información de los navegadores antiguos
 var cadena="!Hola!"
 document.write("<H3> Manejando Cadenas</H3>");
 document.write("La variable 'cadena' contiene ->
"+cadena.bold()+"<BR><BR>");
 document.write("su longitud es de "+cadena.length+" caracteres
<BR>");
 document.write("distribuidos como indica la siguiente tabla:
<BR><BR>");
 document.write("<TABLE BORDER=1><TR>");
 document.write("<TD>Posición</TD>");
  for (i=0; i<cadena.length; i++){
  document.write("<TD>" + i + "</TD>");
 }
 document.write("</TR><TR>");
 document.write("<TD>Contenido</TD>");
 for (i=0; i<cadena.length; i++){
  document.write("<TD><B>" + cadena.charAt(i) + "</B></TD>");
 }
 document.write("</TR></TABLE><BR>");
 document.write("La primera posición en la que aparece '!' es la : ");
 document.write(cadena.indexOf('!'));
 document.write("<BR>La última posición en la que aparece '!' es la :
");
 document.write(cadena.lastIndexOf('!'));
 document.write("<H3>...Hecho</H3>");
// final del comentario -->
</SCRIPT>
```

</CENTER>

</BODY></HTML>

Cabe destacar que el primer elemento de una cadena (string) se encuentra en la posición 0, mientras que el último se encuentra en la posición dada por su tamaño menos 1, es decir, length-1.

2. El objeto Math()

El objeto Math se utiliza para realizar cálculos matemáticos estándar. En lugar de utilizar funciones y constantes matemáticas genéricas, en JavaScript estas funciones se implementan como métodos del objeto Math, es decir, el objeto Math contiene métodos y propiedades que nos permiten representar constantes y funciones matemáticas.

El objeto Math cuenta con las siguientes **propiedades**, todas ellas representan constantes matemáticas y sólo puede accederse a ellas en modo lectura:

- **E:** _Número "e" utilizado como base de los logaritmos neperianos.
- **LN2**: Valor del logaritmo neperiano (logaritmo en base e) de 2.
- **LN10:** Valor del logaritmo neperiano (logaritmo en base e) de 10.
- **LOG2E:** Valor del logaritmo en base 2 de e.
- **LOG10E:** Valor del logaritmo en base 10 de e.
- **PI:** Valor de la constante PI, utilizada para calcular el área de la circunferencia.
- **SQRT1_2:** Valor de la raíz cuadrada de 0,5.
- **SQRT2:** Valor de la raíz cuadrada de 2.

El objeto Math cuenta con los siguientes **métodos**, dentro de los cuales podemos encontrar las funciones trigonométricas básicas, funciones para trabajar con la exponenciación y su inversa, y un conjunto de funciones básicas:

- **abs(número):** Devuelve el valor absoluto de número. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número.

- **acos(número):** Devuelve el arco coseno de número en radianes o NaN. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número dentro del rango -1 a 1 (en otro caso devuelve NaN).
- **asin(número):** Devuelve el arco seno de número en radianes o NaN. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número dentro del rango -1 a 1 (en otro caso devuelve NaN).
- **atan(número):** Devuelve la arco tangente de número en radianes o NaN. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número (en otro caso devuelve NaN).
- **Atan2(coorX,coorY):** Devuelve el ángulo de la coordenada polar correspondiente a las coordenadas cartesianas especificadas. Los argumentos coorX y coorY son números o expresiones o propiedades que se evalúan como un dígito. Representan las coordenadas cartesianas XeY.
- **ceil(número):** Devuelve el número entero que es mayor o igual que número, es decir, redondea número al entero inmediatamente superior. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número.
- **cos(número):** Devuelve el coseno de número en radianes o NaN. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número dentro del rango -1 a 1 (en otro caso devuelve NaN).
- **exp(número):** Devuelve el valor e^{numero}. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número.
- **floor(número):** Devuelve el número entero que es menor o igual que número, es decir, redondea número al entero inmediatamente inferior. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número.
- **log(número):** Devuelve el logaritmo neperiano (logaritmo en base e) de número. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número.
- **max(númerol,número2):** Devuelve el valor mayor de entre número1 y número2. Los rgumentos número 1 y número2 son dígitos o expresiones o propiedades que se evalúan como un número.

- **min(númerol, número2):** Devuelve el valor menor de entre númerol y número2. Los argumentos númerol y número2 son dígitos o expresiones o propiedades que se evalúan como un número.
- **pow(base,exponente):** Devuelve el valor base^{exponente}. Los argumentos base y exponente son dígitos o expresiones o propiedades que se evalúan como un número.
- **random():** Devuelve un número seudoaleatorio entre O y 1.
- **round(número):** Devuelve el valor entero más cercano a número, es decir, redondea número al entero más cercano. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número.
- **sin(número):** Devuelve el seno de número en radianes o NaN. El argumento número es un número o una expresión o propiedad que se evalúa como un número dentro del rango -1 a 1 (en otro caso devuelve NaN).
- **sqrt(número):** Devuelve la raíz cuadrada de número. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número es un dígito o una expresión o
- número.
- **tan(número):** Devuelve la tangente de número en radianes o NaN. El argumento número es un dígito o una expresión o propiedad que se evalúa como un número (en otro caso devuelve NaN).
- **eval();** Evalúa la cadena como una sentencia, en referencia al objeto Math.
- **toString():** Devuelve una cadena que representa al objeto Math.
- **valueOf():** Convierte el objeto a su tipo primitivo (number, boolean,string, o function).

El siguiente ejemplo muestra un ejercicio básico de manejo del objeto Math:

```
<HTML>
<HEAD><TITLE>Math</TITLE>
<SCRIPT LANGUAGE ="JavaScript">
<!-- se oculta la información de los navegadores antiguos
 function calculaEcuacion() {
   var a, b, c, solucion;
```

```
    a=parseFloat(document.miFormulario.a.value);
    b=parseFloat(document.miFormulario.b.value);
    c=parseFloat(document.miFormulario.c.value);
    solucion= (-b + Math.sqrt(Math.pow(b,2)-(4*a*c)))/(2*a);
    document.miFormulario.x1.value=solucion;
    solucion= (-b - Math.sqrt(Math.pow(b,2)-(4*a*c)))/(2*a);
    document.miFormulario.x2.value=solucion;
  }
// final del comentario -->
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
<H1>Manejando el objeto Math...</H1>
<FORM NAME="miFormulario">
<BR><B>Introduce los coeficientes de la</B>
<BR><B>siguiente ecuación de 2º grado: </B><BR><BR>
<INPUT TYPE=TEXT NAME="a" SIZE=3 MAXLENGTH=3> x² +
<INPUT TYPE=TEXT NAME="b" SIZE=3 MAXLENGTH=3> x +
<INPUT TYPE=TEXT NAME="c" SIZE=3 MAXLENGTH=3> = 0<BR>
<BR><B>La soluciones son:</B><BR><BR>
 x = <INPUT TYPE=TEXT NAME="x1" SIZE=5 MAXLENGTH=5> ó
 x = <INPUT TYPE=TEXT NAME="x2" SIZE=5 MAXLENGTH=5>
<BR><BR>
<INPUT TYPE=BUTTON VALUE="Solucionar" NAME="miBoton"
onClick="calculaEcuacion()">
</FORM>
</CENTER>
</BODY>
</HTML>
```

3. El objeto Date()

JavaScript utiliza el objeto Date para proporcionar los métodos necesarios para trabajar con horas y fechas. JavaScript maneja las fechas en milisegundos desde el 1/1/1970 a las 00:00:00 horas, no pudiéndose trabajar con fechas anteriores.

Para representar los meses JavaScript utiliza valores enteros, que van desde el 0 para el mes de Enero al 11 para el mes de Diciembre. Para los días de la semana también utiliza valores enteros, que van desde el 0 para el Domingo hasta el 6 para el Sábado.

Para crear un objeto Date hay que seguir una de las siguientes formas básicas:

1. **nuevoObjetoDate = new Date()** crea un objeto con la fecha y la hora actual.
2. **nuevoObjetoDate =new Date("mes día, año")** crea un objeto con la fecha especificada en el parámetro de tipo cadena.
3. **nuevoObjetoDate=new Date("mes día, año horas ,minutos,segundos")** crea un objeto con la fecha y la hora especificada en el parámetro de tipo cadena.
4. **nuevoObjetoDate = new Date(año, mes, día)** crea un objeto con la fecha especificada en los parámetros de tipo entero.
5. **nuevoObjetoDate = new Date(año, mes, día, horas, minutos, segundos**) crea un objeto con la fecha y la hora especificada en los parámetros de tipo entero.

El objeto Date cuenta con los siguientes **métodos**, dentro de los cuales podemos encontrar las funciones básicas para trabajar con fechas y horas:

- **getDate():** Devuelve el día del mes del objeto Date actual como un entero entre 1 y 31.
- **getDay():** Devuelve el día de la semana del objeto Date actual como un entero entre Oyó.
- **getHours():** Devuelve la hora del objeto Date actual como un entero entre 0 y 23.
- **getMinutes():** Devuelve los minutos de la hora del objeto Date actual como un entero entre 0 y 59.
- **getMonth():** Devuelve el mes del objeto Date actual como un entero entre 0 y 11.

- **getSeconds():** Devuelve los segundos de la hora del objeto Date actual como un entero entre 0 y 59.
- **getTime():** Devuelve la hora completa del objeto Date actual como un entero entre 0 y 23.
- **getTimezoneOffset():** Devuelve la diferencia local y la hora GMT como un entero entre 0 y 23 que representa el número de minutos.
- **getYear():** Devuelve el año del objeto Date actual como un entero de dos dígitos que representa el año menos 1900.
- **parse(fecha):** Devuelve el número de milisegundos desde el 1/1/1970 a las 00:00:00 horas hasta la fecha pasada como parámetro. Esta fecha debe tener uno de los formatos siguiente: "dia, DD mes YYYY HH:MM:SS Zona" ó "mes DD, YYY". Hay que tener en cuanta que parse() es un método estático del objeto Date (pertenece a Date pero no a sus instancias), por tanto, a la hora de llamar a este método se deberá utiliza la sentencia Date.parse().
- **setDate(dia):** Establece el día del mes en el objeto Date actual como un entero entre 1 y 31.
- **setHours(horas):** Establece la hora en el objeto Date actual como un entero entre 0 y 23.
- **setMinutes(minutos):** Establece los minutos de la hora en el objeto Date actual como un entero entre 0 y 59.
- **setMonth(mes):** Establece el mes en el objeto Date actual como un entero entre O y 11.
- **setSeconds(segundos):** Establece los segundos de la hora en el objeto Date actual como un entero entre 0 y 59.
- **setTime(fecha):** Establece la fecha en el objeto Date actual. El parámetro fecha que representa los milisegundos que han pasado desde las 00:00:00 horas del 1/1/1970.
- **setYear(año):** Establece el año en el objeto Date actual. El parámetro año debe ser un entero mayor que 1990.
- **toGMTString():** Devuelve el valor del objeto Date actual como una cadena utilizando las convenciones de Internet (IETF) y con la zona horaria GMT.

- **tolocalString():** Devuelve el valor del objeto Date actual utilizando las convenciones de tiempo locales.
- **UTC(año, mes, día, horas, minutos, segundos):** Devuelve el número de milisegundos desde el 1/1/1970 a las 00:00:00 GMT horas hasta la fecha pasada como parámetro.
- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto Date.
- **toString():** Devuelve una cadena que representa al objeto Date.
- **value0f():** Convierte el objeto a su tipo primitivo {number, boolean, string, o function).

El siguiente ejemplo muestra un ejercicio básico de manejo del objeto Date:

```
<HTML>
<HEAD><TITLE>Date</TITLE>
<SCRIPT LANGUAGE ="JavaScript">
<!-- se oculta la información de los navegadores antiguos
  Date.prototype.cambiaDia=cambiaDia;
  Date.prototype.cambiaMes=cambiaMes;
  Date.prototype.cambiaAnio=cambiaAnio;
  Date.prototype.cambiaFecha=cambiaFecha;

  function cambiaFecha(){
    var fecha="";
    fecha += this.cambiaDia();
    fecha += ", ";
    fecha += this.getDate();
    fecha += " de ";
    fecha += this.cambiaMes();
    fecha += " del ";
    fecha += this.cambiaAnio();
    return fecha;
```

```
}
function cambiaMes(){
    var meses = new Array(12);
   meses[0]= "Enero";
   meses[1]= "Febrero";
   meses[2]= "Marzo";
   meses[3]= "Abril";
   meses[4]= "Mayo";
   meses[5]= "Junio";
   meses[6]= "Julio";
   meses[7]= "Agosto";
   meses[8]= "Septiembre";
   meses[9]= "Octubre";
   meses[10]= "Noviembre";
   meses[11]= "Diciembre";
   return meses[this.getMonth()];
}
function cambiaDia(){
        var dias = new Array(7);
   dias[0]= "Domingo";
   dias[1]= "Lunes";
   dias[2]= "Martes";
   dias[3]= "Miércoles";
   dias[4]= "Jueves";
   dias[5]= "Viernes";
   dias[6]= "Sábado";
   return dias[this.getDay()];
}
function cambiaAnio(){
    return 1900+this.getYear();
}
```

```
// final del comentario -->
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
<SCRIPT LANGUAGE ="JavaScript">
<!-- se oculta la información de los navegadores antiguos
  var fecha1 = new Date();
  var fecha2 = new Date("july 22,1998 12:30:00");
  var fecha3 = new Date("july 22,1997");
  var fecha4 = new Date(1996, 6, 22);
  var fecha5 = new Date(1995, 6, 22, 12,30,00);
  document.write("<H2>Manejando el objeto Date...</H2>");
  document.write("Hoy es " + fecha1.cambiaFecha());
  document.write("<BR><BR>Han pasado ");
  document.write(Date.parse("july 22, 1999"));
  document.write(" milisegundos <BR>");
  document.write("desde las 00:00:00 horas del 1/1/1970<BR>");
  document.write("<BR>Este mismo día en ...<BR><BR>");
  document.write("<TABLE BORDER=1 CELLPADDING=4>");
  document.write("<TR ALIGN=CENTER><TD>el año...</TD>");
  document.write("<TD>" + fecha2.getYear() + "</TD>");
  document.write("<TD>" + fecha3.getYear() + "</TD>");
  document.write("<TD>" + fecha4.getYear() + "</TD>");
  document.write("<TD>" + fecha5.getYear() + "</TD></TR>");
  document.write("<TR ALIGN=CENTER><TD>era un...</TD>");
  document.write("<TD>" + fecha2.cambiaDia() + "</TD>");
  document.write("<TD>" + fecha3.cambiaDia() + "</TD>");
  document.write("<TD>" + fecha4.cambiaDia() + "</TD>");
  document.write("<TD>" + fecha5.cambiaDia() + "</TD></TR>");
  document.write("</TABLE>");
```

```
  document.write("<H2>...Hecho</H2>");
// final del comentario -->
</SCRIPT>
</CENTER>
</BODY>
</HTML>
```

En el código se definen 5 objetos de tipo Date de las 5 posibles formas. Después se manejan dichos objetos para obtener información básica de ellos. Se definen cuatro funciones que nos permiten pasar a castellano la información suministrada por los objetos: dichas funciones se asignan como nuevas propiedades del objeto Date haciendo uso de la propiedad prototype.

4. El objeto Bolean().

El objeto Boolean nos permite crear objetos booleanos, es decir, nos permite manejar valores booleanos como objetos. También permite transformar datos no booleanos en datos booleanos. Para crear objetos de tipo Boolean en JavaScript utilizamos el constructor new y el objeto constructor Boolean(), con el siguiente formato:

NuevoBooleano = new Boolean(valor);

Cuando no se proporciona el argumento valor, o éste es 0, una cadena de caracteres vacía o el valor false, el nuevo objeto toma el valor false. Cuando se proporciona un argumento distinto de 0, la cadena vacía o éste toma el valor true, el nuevo objeto toma el valor true.

Los objetos de tipo Boolean tienen la siguiente **propiedad:**

- **prototype:** Nos permite asignar nuevas propiedades al objeto Boolean .

El objeto Boolean posee sólo tres **métodos**:

- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto Boolean.

- **toString():** Devuelve una cadena que representa al objeto Boolean.
- **value0f():** Convierte el objeto a su tipo primitivo (number, boolean,string, o function).

5. El objeto Number()

El objeto Number nos proporciona propiedades y métodos básicos para trabajar con valores numéricos. Es especialmente útil cuando necesitamos manejar algunas de sus propiedades. Para crear objetos de tipo Number en JavaScript utilizamos el constructor new y el objeto constructor Number(), con el siguiente formato:

NuevoNumero = new Number(valor);

Donde el parámetro valor es el valor inicial del nuevo objeto de tipo Number que se ha creado.

Los objetos de tipo Number tienen las siguientes **propiedades:**

- **MAX_VALUE:** Es el mayor número representable.
- **MIN_VALUE:** Es el menor número representable.
- **NaN**: Valor de Not-A-Number devuelto cuando un elemento no es un número.
- **NEGATIVE_INFINITY:** Valor negativo infinito devuelto cuando se produce un overflow.
- **POSITIVE_INFINITY:** Valor positivo infinito devuelto cuando se produce un overflow.
- **prototype:** utilizado para asignar nuevas propiedades a un objeto de tipo Number.

El objeto Number posee sólo tres **métodos:**

- **eval():** Evalúa la cadena como una sentencia, en referencia al objeto Number.
- **toString():** Devuelve una cadena que representa al objeto Number.
- **valueOf():** Convierte el objeto a su tipo primitivo (number, boolean,string, o function).