# UNIDAD 5 - OBJETOS

## 5.1 - OBJETOS EN JAVASCRIPT

Los objetos son los principales protagonistas del paradigma de la programación orientada a objetos. Son una evolución de otro tipo estructurado denominado registro (estructuras en algunos lenguajes). Los objetos representan entidades cuya complejidad hace que no sea suficiente un valor para almacenar toda la información necesaria. Por este motivo, los registros o estructuras son conjuntos de datos de diferente tipo relacionados entre sí. Por ejemplo, los datos de una persona (nombre, apellidos, fecha de nacimiento, dirección, etc.) se pueden agrupar en un registro o estructura. El paradigma orientado a objetos incorpora a esos datos estructurados, las funciones (llamadas métodos) que manipulan esos datos, haciendo que los datos y los procedimientos conformen un ente llamado objeto. En Javascript todas las variables hacen referencia a objetos y en realidad todo son objetos incluyendo los tipos de datos, las estructuras e incluso las propias funciones. En Javascript hay numerosos objetos predefinidos y el usuario-programador puede definir los que necesite. Entre los objetos predefinidos asociados a elementos del navegador destacan las ventanas, los formularios, los controles, y en general todos los elementos que permiten obtener la experiencia de la navegación.

### 5.1.1 -OPERADOR NEW

Para crear un objeto en Javascript se utiliza el mecanismo declarativo que se ha visto en los temas anteriores o bien mediante el operador ***new***. Con carácter general se recurre al método declarativo cuando se trabaja con tipos simples de datos y cadenas de caracteres, recurriendo al método más formal basado en el operador ***new*** para el resto de los objetos. Por ejemplo, el método declarativo para una cadena de caracteres sería

***miCadena="";*** mientras que el método basado en el operador new quedaría así ***miCadena = new String();***

Javascript no posee todas las características de los lenguajes orientados a objeto ni es objetivo de este curso enseñar a programar utilizando este potente y práctico paradigma. Sin embargo es importante conocer algunos aspectos de los objetos ya que la mayor parte de los mecanismos están ligados a los mismos. Y todo ello sin olvidar que existen muchos objetos predefinidos que resuelven procesos sin necesidad de programar nada. Entre los objetos más relevantes destacan los siguientes: Array, Boolean, Date, Function, Math, Number y String entre otros.

Para crear objetos a medida de las necesidades del programador hay que establecer en primer lugar los datos y a continuación los procedimientos que manipulan esos datos. En la jerga del paradigma los primeros reciben el nombre de atributos/propiedades mientras que los segundos se denominan métodos. Por ejemplo, para definir un tipo de objeto que represente los datos de una cuenta bancaria simplificada y crear un objeto concreto se procede de la siguiente forma:

```
function cuenta_bancaria( numero, titular, saldo)
{
        this.numero = numero;
        this.titular = titular;
        this.saldo = saldo;
}

var mi_cuenta = new cuenta_bancaria (123,"Pedro García", 1500);
```

Como se puede observar el objeto llamado ***mi_cuenta*** se crea con el operador new seguido del nombre del tipo de objeto y poniendo entre paréntesis los valores que se desean tomen los atributos del objeto. En el interior de la función cuyo nombre coincide con el tipo de objeto aparece el operador ***this*** que hacer referencia al propio objeto que se está creando. El lector en este punto puede simplemente recordar la sintaxis sin pararse a analizar en detalle el motivo de su uso. La flexibilidad de JavaScript permite incluso crear nuevos atributos para un objeto previamente creado aunque es recomendable no hacerlo para evitar programas complejos de interpretar por una persona.

Para acceder a cada uno de los atributos de un objeto se utiliza la notación punto que consiste en escribir primero el nombre del objeto seguido de un punto y la propiedad específica. Por ejemplo, para mostrar en pantalla una alerta (ventana emergente y modal) con el saldo de la cuenta se escribirá ***alert (mi_cuenta.saldo);***

## 5.2 - ARRAYS

En los lenguajes de programación estructurados, los arrays constituían un tipo concreto de estructuras de datos. En la actualidad, con la incorporación del paradigma orientado a objetos, la mayor parte de los lenguajes que siguen este paradigma conciben los arrays como objetos.

### 5.2.1 -CREACIÓN DE UN ARRAY.

Para crear un array en Javascript se utiliza el siguiente formato:

***var NombreDelArray = new Array (NumeroDeElementos)***

NombreDelArray es el nombre o identificador del array y NumeroDeElementos es un número que indica el número de elementos que contendrá.

Para crear el Array Alumnos que contendrá 5 elementos, escribiremos:

```
var Alumnos = new Array (5);
```

Ahora tenemos creada la estructura y las posiciones de memoria están reservadas y disponibles, aunque vacías, porque todavía no hemos introducido en ellas ningún valor.

Cuando no conocemos previamente el número de elementos que va a tener un array no se definirá ningún número en el parámetro ***NumeroDeElementos***. De esta forma el Array se dimensionará automáticamente al número de elementos que se vayan introduciendo.

***var NombreDelArray = new Array ()***

## 5.2.2 -ASIGNACIÓN DE ELEMENTOS EN UN ARRAY.

Un array es, estructuralmente hablando, un conjunto de variables del mismo tipo identificadas por un único nombre y diferenciadas entre sí por la posición relativa que ocupan. Como consecuencia de ello el operador de asignación se puede utilizar de igual forma que se hace con variables normales. No obstante hay que señalar siempre a qué elemento en concreto nos referimos del array. Por ejemplo, para asignar al elemento que ocupa la posición 3 un valor hay que poner como índice el valor 2 (de nuevo recuerde que los arrays se numeran comenzando por el cero): ***Alumnos[2] = "PEPE";***

**Recorrido de los elementos de un Array.**

Las variables numéricas enteras, en combinación con estructuras de control repetitivas se utilizan frecuentemente para recorrer los elementos de un array. En efecto, supongamos la siguiente estructura:

```
for (i = 0; i < 5; i = i + 1)
{
        document.write(Alumnos[i]);
}
```

El bucle for se ejecutará 5 veces con la instrucción document.write. Para cada una de estas ejecuciones la variable i tomará uno de los valores correspondientes al índice del Array (0, 1, 2, 3 y 4). En los arrays está disponible la propiedad length que devuelve el número de elementos que tiene un array (incluyendo los elementos vacíos, si hubiese). Su formato genérico es: ***NombreDelArray.length***

Utilizando esta propiedad podemos escribir una estructura que recorrería cualquier array completo:

```
for (i = 0; i < NombreDelArray.length; i = i + 1)
{
Tratamiento_del_elemento_NombreDelArray[i]);
}
```

Veamos un ejemplo completo de creación del Array, inicialización de los elementos e impresión de los mismos.

```
<HTML>
<HEAD><TITLE>Manipulando Arrays</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
/* Ejemplo Arrays 1*/
var Alumnos = new Array (5);
Alumnos[0] = "MIGUEL";
Alumnos[1] = "ELENA";
Alumnos[2] = "JORGE";
Alumnos[3] = "CRISTINA";
Alumnos[4] = "IGNACIO";
for ( i = 0; i < 5; i++)
{
        document.write(Alumnos[i] + "*");
}
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 25 – Manipulando arrays***

El resultado de la ejecución de este programa será:

MIGUEL*ELENA*JORGE*CRISTINA*IGNACIO*

Para recorrer los elementos de un array existe el bucle for in. Este bucle permite recorrer todos los elementos del array sin necesidad de saber su número.

```
var miArr = [ 'uno', 'dos', 'tres'];
 for( var elemento in miArr ){
  document.write( miArr[elemento] );
}
```

## 5.2.3 -INTRODUCCIÓN MANUAL DEL CONTENIDO EN UN ARRAY.

Hasta el momento hemos trabajado con un array cuyos datos se introducen directamente desde el código JavaScript. Pero en ocasiones necesitaremos que sea el usuario quien introduzca los elementos del array. A continuación estudiaremos un programa en el que el usuario introduce los datos que se guardan en el array de manera secuencial (cada elemento a continuación del anterior)

```
<HTML><HEAD><TITLE>Introducción manual de datos en un Array</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
/* Ejemplo Arrays Con valores introducidos por el usuario*/
var Alumnos = new Array (5);
for ( i = 0; i < 5; i++)
{
Alumnos[i] = prompt("Introduce nombre del alumno "+i, "");
}
for ( i = 0; i < 5; i++)
{
document.write(Alumnos[i] + "*");
}
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 26 – Introducción manual de datos en un array***

JavaScript también permite crear un array e introducir los elementos simultáneamente desde el código declarando el nombre y enumerando a continuación en una lista los elementos:

```
var Alumnos = new Array ("MIGUEL", "ELENA", "JORGE","CRISTINA","IGNACIO");
```

El array se dimensionará implícitamente en función del número de elementos. Por su parte, los elementos definidos así se asociarán con el índice según la posición que ocupan la lista (el primero será el 0, el segundo el 1, y así sucesivamente).

### 5.2.4 -BÚSQUEDA EN UN ARRAY.

Una de las operaciones más comunes a la hora de manipular información contenida en arrays es realizar búsquedas. Para ello o bien se conoce la posición que ocupa el elemento o bien se busca el elemento por su contenido. La primera no plantea ningún problema ya que si conocemos el índice el acceso al elemento correspondiente es automático. (Por ejemplo, si buscamos el elemento 3 accedemos a él como Alumnos[2] –recuerde de nuevo el hecho de que el primer elemento ocupa el índice cero-). La segunda implica un recorrido del array comparando uno a uno cada elemento con el valor que se busca hasta obtener dicho valor o llegar al final sin obtener un resultado satisfactorio. En este caso hay que emplear algunas líneas de programa:

```
...
Vbusca = ... // Vbusca es la variable que suponemos
// contiene el valor a buscar
Ultimo = NombreDelArray.length -1
i = 0;
while (NombreDelArray[i] != Vbusca && i < Ultimo)
{
i = i + 1;
}
```

La salida del bucle se produce por una de las siguientes circunstancias: ha llegado al último elemento; o bien, ha encontrado en valor buscado. Deberemos, por tanto, comprobar si realmente ha encontrado lo que buscaba o no.

```
if (NombreDelArray[i] == Vbusca)
{
        ...Tratamiento en caso de encontrar el elemento.
}
else
{
        ...Tratamiento en caso de NO encontrar el elemento.
}
```

El siguiente ejemplo pide introducir un nombre de alumno por teclado, si el nombre de alumno existe en la lista se visualizará el mensaje “Encontrado”, si no existe se visualizará el mensaje de “No encontrado”

```
<HEAD><TITLE>Búsqueda en un Array</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
/************ Ejemplo búsqueda en Arrays**************/
var Alumnos = new Array ("MIGUEL", "ELENA", "JORGE","CRISTINA","IGNACIO", "ALICIA");
Ultimo = Alumnos.length - 1;
Vbusca ="";
i= 0;
/************ Entrada del nombre a buscar ************/
Vbusca = prompt("Introduce el nombre del alumno en mayúsculas", " ");
/******************** Búsqueda **********************/
i = 0;
while ( Alumnos[i] != Vbusca && i < Ultimo)
{
        i = i + 1;
}
/************** Comprobación y visualización *********/
if (Alumnos[i] == Vbusca)
{
        alert("Encontrado el alumno: " + Vbusca +" con el número: " + i);
}
else
{
        alert("No encontrado el alumno: " + Vbusca);
}
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 27 – Búsqueda en un array***

## 5.2.5 -PROPIEDADES DE LOS ARRAYS

**Length:** Devuelve la longitud del array, es decir, el número de elementos que puede almacenar. Su uso es muy simple:

```
var vector = new Array(50);
longitud = vector.length; /* longitud valdría 50 */
```

**prototype:** Permite agregar al objeto array nuevas propiedades y métodos. Por ejemplo, para añadir una propiedad llamada denominación se procedería de la siguiente forma:

```
Array.protoype.denominacion = null;
dias = new Array ('lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes');
dias.denominacion = "Dias laborables";
```

## 5.2.6 -MÉTODOS DE LOS ARRAYS

Todos los ejemplos están extraídos de http://www.w3schools.com/jsref/jsref_obj_array.asp página en la que además puede consultar muchos más métodos y sus ejemplos con ejecutables y código disponible.

| Función / Operación | Ejemplo | Resultado |
|---|---|---|
| **concat(objArray):** Devuelve un array resultado de concatenar el objeto invocado con el objeto que se le pasa como argumento. | <HTML><br><HEAD><br><SCRIPT languaje="Javascript"><br>function myFunction()<br>{<br>var hege = ["Cecilie", "Lone"];<br>var stale = ["Emil", "Tobias", "Linus"];<br>var kai = ["Robin"];<br>var children = hege.concat(stale,kai);<br>alert (children);<br>}<br></SCRIPT><br></HEAD><br><BODY><br><SCRIPT languaje="Javascript"><br>myFunction();<br></SCRIPT><br></BODY><br></HTML> | Cecilie,Lone,Emil,Tobias,Linus,Robin |

| | | |
|---|---|---|
| **join():** Convierte los elementos de un array en una cadena separados por el carácter que se le indique. El separador por defecto es la coma. | function myFunction()<br>{<br>var fruits = ["Banana", "Orange", "Apple", "Mango"];<br>var x=document.getElementById("demo");<br>var y=fruits.join();<br>alert (y);<br>} | Banana,Orange,Apple,Mango |
| **reverse():** Invierte el orden de los elementos de un array en el propio array sin crear uno nuevo. | function myFunction()<br>{<br>var fruits = ["Banana", "Orange", "Apple", "Mango"];<br>var y=fruits.reverse();<br>alert (y);<br>} | Mango,Apple,Orange,Banana |
| **slice(ini, fin):** Extrae parte de un Array devolviéndolo en un nuevo objeto array. | function myFunction()<br>{<br>var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];<br>var citrus = fruits.slice(1,3);<br>alert (citrus);<br>} | Orange,Lemon |
| **sort(rutord):** Ordena alfabéticamente los elementos de un objeto Array. Opcionalmente admite un argumento adicional que será una función para determinar el orden. La función tiene dos argumentos y devolverá un valor negativo si el primer argumento es menor que el segundo, cero si son iguales y un valor positivo si el primer argumento es mayor que el segundo. En castellano esto es necesario si queremos que la ñ y vocales acentuadas figuren en su lugar. | function myFunction()<br>{<br>var fruits = ["Banana", "Orange", "Apple", "Mango"];<br>fruits.sort();<br>alert (fruits);<br>} | Apple,Banana,Mango,Orange |

## 5.3 - REFERENCIA COMPLETA DE OBJETOS

Una relación muy completa de objetos con sus propiedades, métodos y ejemplos ejecutables se puede consultar en la siguiente Web:

http://www.w3schools.com/jsref/default.asp

Si visita la referencia anterior podrá encontrar una clasificación muy interesante sobre los objetos predefinidos en Javascript. El conocimiento y dominio de los mismos permitirá diseñar aplicaciones muy versátiles con muy poco esfuerzo. En primer lugar aparecen enlaces a los diferentes objetos JavaScript: *Array object, Boolean object, Date object, Math object, Number object, String object, RegExp object*, así como la relación de propiedades y funciones globales: *Global properties and functions*. Con ellos se pueden incorporar y manipular de forma sencilla datos y estructuras de datos en los programas. A pesar de que el lector pueda entender que el abanico de opciones puede llegar a ser complejo, la realidad demuestra todo lo contrario. La existencia de todos estos objetos agiliza el desarrollo de aplicaciones. En Javascript con un método de un objeto se pueden hacer cosas que requerirían muchas líneas de código en otros lenguajes de programación.

A continuación se incluyen enlaces a los objetos del navegador: *Window object, Navigator object, Screen object, History object, Location object*. Estos objetos permiten que desde un programa Javascript se tenga control sobre los objetos del navegador y por lo tanto permiten controlar la experiencia de navegación. Además hay enlaces a los objetos DOM que permiten manipular ficheros XML en general y por tanto también el propio HTML: *DOM Node, DOM NodeList, DOM NamedNodeMap, DOM Document, DOM Element, DOM Attr*. Por último referencias a objetos DOM HTML: *Document object, Event object, HTMLElement object, Anchor object, Area object, Base object, Body object, Button object, Form object, Frame/IFrame object, Frameset object, Image*

*object, Input Button object, Input Checkbox object, Input File object, Input Hidden object, Input Password object, Input Radio object, Input Reset object, Input Submit object, Input Text object, Link object, Meta object, Object object, Option object, Select object, Style object, Table object, td / th object, tr object, Textarea object* que constituyen un control completo sobre los elementos HTML que conforman la visualización de una aplicación Web.

## 5.4 - LOS OBJETOS DEL NAVEGADOR

Los objetos asociados al navegador responden a una jerarquía que permite referenciar a cada uno de los objetos. Existen dos jerarquías de objetos, una la del objeto window formada por todos los elementos presentes en la visualización de un documento HTML, y otra la del objeto navigator relacionado con las características del navegador. La mayoría de los objetos del navegador tienen una correspondencia con las etiquetas HTML, así la etiqueta **<FRAME>** se corresponde con el objeto *Frame*, la etiqueta **<FORM>** con el objeto *Form*, la etiqueta **<A HREF=" ">** con el objeto *Link*, la etiqueta **<INPUT TYPE=BUTTON>** con el objeto *Button*, la etiqueta **<BODY>** con el objeto *Document*, etc, etc. Sólo los objetos *window, history y locati*on no tienen una correspondencia directa con las etiquetas HTML.

Según esta jerarquía, podemos decir por ejemplo que el objeto caja de texto *text* es un objeto que está dentro del objeto *form* que a su vez está dentro del objeto *document* y que este está dentro del objeto *window*. Para hacer referencia a la caja de texto, tendremos que escribir: ***window.document.form.text.*** En la mayoría de los casos podemos ignorar la referencia a la ventana actual (*window*), pero será necesaria cuando estemos utilizando múltiples ventanas, o cuando usemos frames (actualmente en desuso).

El uso de arrays como elementos dentro de la estructura jerárquica dentro del modelo de objetos de JavaScript es muy importante. Proporcionan una manera alternativa para referenciar a los elementos. Así se puede referenciar a un objeto por su posición dentro del array que lo contiene. Los siguientes arrays son los más utilizados para referenciar los componentes de un documento:

- Frames[posición]: contiene los diferentes marcos del documento.
- Links[posición]: contiene los diferentes enlaces externos de un documento.
- Images[posición]: contiene las diferentes imágenes de un documento.
- Forms[posición]: contiene los diferentes formularios de un documento.
- Elements[posición]: contiene los diferentes elementos de un formulario.
- options[posición]: contiene las diferentes opciones de un objeto select.

En el siguiente ejemplo se muestran los nombres de algunos de los objetos del documento HTML utilizando la notación de array. La función objetos() se encarga de visualizar los nombres de los objetos, la función ver(dato) se encarga de visualizar el dato en pantalla y al pulsar en el botón del formulario se visualiza el contenido del texto escrito en la caja de texto, esta misión la realiza la función vertexto(). Observe que para obtener el nombre de un objeto se utiliza la palabra name. Por ejemplo con la sentencia document.images[0].name se obtiene el nombre de la primera imagen del documento HTML.

**Objeto window:** representa la ventana del navegador o cualquiera de sus marcos, es el de mayor jerarquía. Por cada etiqueta **BODY** o **FRAMESET** se genera un objeto ***window***.

**Objeto document:** es el que tiene el contenido de toda la página que se está visualizando. Esto incluye el texto, imágenes, enlaces, formularios. Gracias a este objeto vamos a poder añadir dinámicamente contenido a la página, o hacer cambios, según nos convenga.

**Objeto form:** contiene los elementos necesarios para generar formularios. Los formularios se agrupan en un array dentro de document. Cada elemento de este array es un objeto de tipo ***form***.

**Objeto frame:** un objeto frame se comporta igual que un objeto window y contiene sus mismas propiedades y métodos. No obstante este mecanismo está en desuso por cuestiones de accesibilidad.

**Objeto history:** contiene la información de las URL que el usuario ha visitado desde esa ventana.

**Objeto location:** contiene información sobre la URL actual.

```
<html>
<head>
<title>Identificando objetos</title>
<SCRIPT LANGUAGE="JavaScript">
function ver(dato)
{  document.write(dato); }
function objetos()
{
ver("<BR><CENTER>");
ver("Nombre de la primera imagen:"+document.images[0].name+ "<BR>"); ver("Nombre de la
segunda imagen:"+document.images[1].name+ "<BR>"); ver("Número de
imágenes:"+document.images.length+ "<BR>"); ver("Nombre del segundo
vínculo:"+document.links[1].name+ "<BR>");
ver("Número de elementos del formulario:"+document.forms[0].elements.length+ "<BR>");
}
function vertexto()
{ ver("Contenido del texto:"+document.forms[0][0].value+ "<BR>"); }
</SCRIPT>
</head>
<body>
<p align="center">
<img border="0"
src="http://t2.gstatic.com/images?q=tbn:ANd9GcQ_FVeoXTQd7fc2fqxrLGWD9COxnH2fl9iajoMNy
bMP2b2ghEv2yA" width="21" height="21" NAME ="ALFI1">Alfi1 
<a href="http://www.mec.es" NAME ="MEC"> --MEC-- </a>
<img border="0"
src="http://t2.gstatic.com/images?q=tbn:ANd9GcTyDjBaGR_IhifCmYMdAI5dSYPTgBkH4V4Sbxzxp
pKzEPWUX1Pn" width="21" height="21" NAME ="ALFI3">Alfi3
</p>
<p align="center">
<a href="http://www.aulamentor.es/" NAME ="EJEMPLO">Aula Mentor</a>  
<img border="0"
src="http://t0.gstatic.com/images?q=tbn:ANd9GcRteJuAuPRYGBbRALYjlxVwcY_TtG2qWDesqsjP5
BLf1iWyVlIObQ" width="21" height="21" NAME ="ALFI2"> Alfi2 
<center>
<table border="0" cellpadding="0" cellspacing="0" bgcolor="#FFFF99">
<tr><td>
<p align="center"><b>FORMULARIO</b><p>
<form NAME ="FORMULARIO">
TEXTO<input type="text" name="TEXTO" size="20">
<input type="submit" value="Ver texto" name="BOTON" onclick="vertexto()">
</form>
</td></tr></table></center>
<SCRIPT LANGUAGE="JavaScript">
objetos();
</script>
</body>
</html>
```

***Ejemplo 28 – Identificación de objetos***

## 5.5 - LOS OBJETOS PREDEFINIDOS.

Estos objetos están relacionados con el propio lenguaje y nos permitirán manejar nuevas estructuras de datos y utilidades. Algunos ya se han utilizado anteriormente como el objeto Array, Date, Function, etc. El siguiente ejemplo muestra el uso de los objetos Array, String y Math con algunos de sus métodos: cad es un objeto String, nombres un objeto Array y a numero le aplicamos algunos métodos del objeto Math:

```
<HTML>
<HEAD>
<TITLE>Objetos predefinidos</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function probando() {
var cad = "Objetos predefinidos";
var nombres=new Array("Juan","Ana","Julio","Maria","Alicia");
var numero=36;

document.write("Longitud de la cadena: "+cad.length+"<BR>");
document.write("Mayúscula: "+cad.toUpperCase()+"<BR>");
document.write("De color rojo: "+cad.fontcolor("#FF0000")+"<P>");

document.write("Antes de ordenar el array: " +nombres.join(', ')+"<BR>");
document.write("Nombres ordenados: "+nombres.sort()+"<BR>");
document.write("Ordenados en orden inverso: " +nombres.sort().reverse()+"<P>");

document.write("Raiz cuadrada de 36: " +Math.sqrt(numero)+"<BR>");
document.write("36 elevedo a 5: " + Math.pow(numero,5)+"<BR>");
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
probando();
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 29 – Objetos predefinidos***

## 5.6 - PROPIEDADES Y MÉTODOS DE LOS OBJETOS DEL NAVEGADOR.

### 5.6.1 -EL OBJETO WINDOW.

En el objeto ventana es donde acontecen todas las operaciones del documento. Las propiedades y métodos se resumen en la siguiente tabla, incluiremos también los manejadores de eventos aunque en la siguiente unidad se hablará más de ellos.

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **closed, default, Status, frames, history, length, location, name, parent, opener, self, status, top.** | **alert(mensaje), open(), close(), confirm(mensaje), prompt(), moveBy(x,y), moveTo(x,y), print(), setTimeout(), resizeBy(x,y), resizeTo(ancho,alto), scroll(x,y), scrollBy(x,y), scrollTo(x,y), clearInterval() , setInterval(), clearTimeout() , setTimeout()** | **onLoad, onUnload** |

Para crear una ventana: ***nuevaventana = window.open()***. Para acceder a las propiedades o métodos: ***window.propiedad window.método() self.propiedad, self.método()***

### 5.6.2 -PROPIEDADES DEL OBJETO WINDOW

**closed:** Válida a partir de JavaScript 1.1 (a partir de Netscape 3 e Internet Explorer 4). Es un valor booleano que nos dice si la ventana está cerrada (closed = true) o no (closed = false).

**defaultStatus:** Contiene el texto por defecto que aparece en la barra de estado del navegador.

**frames:** Es un array que representa a los objetos frame contenidos en la ventana (frames[0], frames[1], ...). El orden de aparición (frame[0],...) es el orden en que se definen en el documento HTML.

**history:** Se trata de un array que representa las URLS visitadas por la ventana (están almacenadas en el historial).

**length:** Número de frames que contiene la ventana actual.

**location:** Cadena con la URL de la barra de dirección.

**name:** Nombre de la ventana, o del frame actual.

**opener:** Es una referencia al objeto window que abrió la ventana si la ventana fue abierta usando el método open().

**parent:** Referencia al objeto window que contiene el frameset (padre de esta ventana).

**self:** Es un nombre alternativo de la ventana actual.

**status:** Cadena con el mensaje que tiene la barra de estado.

**top:** Nombre alternativo del objeto window padre de esta ventana.

```
<SCRIPT LANGUAGE="JavaScript">
window.name="ventanita";
window.status="Esto es un ejemplito";
document.write("¿Está la Ventana cerrada? "+window.closed+"<br>");
document.write("Nombre de ventana: "+window.name+"<br>");
document.write("Mensaje de la barra de estado: "+window.status+"<br>");
document.write("Valor de self: "+window.self.name+"<br>");
document.write("Valor de top: "+window.top.name+"<br>");
document.write("Valor de location: "+window.location+"<br>");
</SCRIPT>
```

**Ejemplo:** Crear una ventana dinámicamente según los datos establecidos por el usuario a través de un formulario.

***open(URL,nombre,características):*** Abre la URL que le pasemos como primer parámetro en la ventana cuyo nombre se indica en el segundo parámetro 'nombre'. Si esta ventana no existe, abrirá una nueva ventana con las características indicadas en el tercer parámetro.

| | | |
|---|---|---|
| **Toolbar** | yes,no,1,0 | Tendrá (yes,1) o no (no,0) barra de herramientas. |
| **location** | yes,no,1,0 | Tendrá campo de localización o no. |
| **directories** | yes,no,1,0 | Tendrá botones de dirección o no. |
| **status** | yes,no,1,0 | Tendrá barra de estado o no. |
| **menubar** | yes,no,1,0 | Tendrá barra de menús o no. |
| **scrollbars** | yes,no,1,0 | Tendrá barras de desplazamiento o no. |
| **resizable** | yes,no,1,0 | Podrá ser cambiada de tamaño (con el ratón) o no. Devuelve el ancho de la ventana en pixels |
| **width** | Nº de pixels | |
| **height** | Nº de pixels | Devuelve el alto de la ventana en pixels. |
| **left** | Nº de pixels | Devuelve la distancia en pixels desde el lado izquierdo de la pantalla a la que se debe colocar la ventana. |
| **top** | Nº de pixels | Devuelve la distancia en pixels desde el lado superior de la pantalla a la que se debe colocar la ventana |

El método ***close()*** cierra la ventana actual. La función ***CrearVentana()*** crea la ventana y la función ***CerrarVentana()*** cierra las dos ventanas, la ventana actual y la ventana creada. Las sentencia siguiente: ***if(ventana && !ventana.closed) ventana.close();*** comprueba si la ventana creada existe y además que no esté cerrada, si se cumplen las condiciones se cierra dicha ventana. Para obtener el valor de un objeto utilizamos la palabra value. Por ejemplo con la sentencia ***document.FORMULARIO[0].value*** obtenemos el valor introducido en el primer elemento del formulario cuyo nombre es FORMULARIO.

```
<HTML>
<HEAD>
<TITLE>OBJETOS - Ejemplo  - VENTANAS</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var ventana;

function CrearVentana()  {
var opciones="left=100,top=100,";

opciones=opciones + "width=" + document.FORMULARIO[0].value +",";
opciones=opciones + "height=" + document.FORMULARIO[1].value ;

ventana = window.open("","",opciones); //crea la ventana
ventana.document.write("ESTA ES LA VENTANA QUE SE HA CREADO");
}

function CerrarVentana()  {
if(ventana && !ventana.closed)
ventana.close(); //cierra la ventana creada
window.close();//cierra la ventana actual
}
</SCRIPT>
</HEAD>
<BODY>
<p align="center"><b>INTRODUCE EL ANCHO Y ALTO DE LA VENTANA:</b></p>
<center>
<table border="0" cellpadding="0" cellspacing="0">
<tr>
<td bgcolor="#FFFF66">
<form  name="FORMULARIO" >
<p><b>ANCHO VENTANA:</b><input type="text" name="ANCHO" size="10"></p>
<p><b>ALTO VENTANA:</b> <input type="text" name="ALTO" size="10"></p>
<input type="button" value="Crear Ventana" name="B1" onclick="CrearVentana()">
<input type="button" value="Cerrar Ventana" name="B2" onclick="CerrarVentana()">
</form>
</td>
</tr>
</table>
</center>
</FORM>
</BODY>
</HTML>
```

***Ejemplo 30 –Manipulación de objeto ventana***

**Ejemplo:** Supongamos que se definen tres botones en un formulario, dos de los botones llaman a otro documento HTML que se abre en una ventana diferente y el tercer botón cierra la ventana actual. El código del formulario es el siguiente:

```
<form name="FORMULARIO" >
<input type="button" value="Ejemplo 5" name="B1" onclick="llamada(0)">
<input type="button" value="Ejemplo 4" name="B2" onclick="llamada(1)">
<input type="button" value="Cerrar Ventana" name="B2" onclick="CerrarVentana()">
</form>
```

La función ***llamada()*** se encargará de cargar los documentos HTML en otra ventana. Esta función recibe un número indicando el documento a cargar (si es 0 se carga un documento y si es 1 se carga otro), sus nombres están cargados en un array llamado paginas. El método ***window.open()*** crea la ventana donde se cargarán los documentos HTML. Para cargar la página en la ventana creada escribimos la siguiente sentencia: ***nuevo.location.href = paginas[p];*** donde ***href*** es una propiedad del objeto ***location***, nuevo es la nueva ventana creada, la sentencia hace que se carge en la nueva ventana la URL contenida en el array ***paginas[p]***.

Todos los objetos window y frame tienen asociado un objeto location. Este objeto permite modificar la URL de la página para cambiar a una nueva URL. El código de la función ***llamada()*** es el siguiente:

```
function llamada(p)
 {
var nuevo=null;
var paginas=new Array("Ejemplo5_5.htm","Ejemplo5_4.htm");
nuevo = window.open('', 'VE','width=400,height=390,top=50,left=100');
nuevo.location.href = paginas[p];
}
```

Se creará una única ventana de nombre VE, donde se visualizarán los documentos HTML.

### 5.6.3 -MÉTODOS PARA MOVER Y REDIMENSIONAR VENTANAS

**resizeBy(x,y):** Aumenta/disminuye el tamaño actual de la ventana en el número de pixeles indicados en x e y.

**resizeTo(ancho,alto):** Ajusta el tamaño de la ventana a los valores indicados por la ventana.

**moveBy(x,y):** Mueve la ventana actual el número de pixels especificados por (x,y).

**moveTo(x,y):** Mueve la ventana actual a las coordenadas (x,y).

**scroll(x,y):** Hace que en la esquina superior izquierda aparezca la ventana a partir de las coordenadas indicadas por x e y.

**scrollBy(x,y):** Desplaza la ventana actual el número de pixels especificado por (x,y).

**scrollTo(x,y):** Desplaza la ventana actual a las coordenadas especificadas por (x,y).

**Ejemplo:** Este ejemplo mueve la ventana actual (moveTo) o la redimensiona (resizeTo) según los datos introducidos en los campos del formulario.

```
function llamada(p)
 {
var nuevo=null;
var paginas=new Array("Ejemplo5_5.htm","Ejemplo5_4.htm");
nuevo = window.open('', 'VE', 'width=400,height=390,top=50,left=100');
nuevo.location.href = paginas[p];
}
```

El código del formulario es:

```
<form name="FORMULARIO" >
<p><b>Coordenada X:</b><input type="text" name="X" size="10"></p>
<p><b>Coordenada Y:</b> <input type="text" name="Y" size="10"></p>
<input type="button" value="Mover ventana" name="B1"
onclick="MoverVentana()">
<input type="button" value="Redimensionar" name="B2"
onclick="Redimensionar()">
</form>
```

El código de las funciones es:

```
<SCRIPT LANGUAGE="JavaScript">
var x,y;
function MoverVentana() {
x= document.FORMULARIO[0].value ;
 y = document.FORMULARIO[0].value;
window.moveTo(x,y);
}
function Redimensionar() {
x= document.FORMULARIO[0].value ;
 y = document.FORMULARIO[0].value;
window.resizeTo(x,y) ;
}
</SCRIPT>
```

## 5.6.4 -OTROS MÉTODOS

**alert(mensaje):** Muestra el mensaje 'mensaje' en un cuadro de diálogo.

**blur():** Elimina el foco de la ventana y dispara el evento onBlur.

**clearInterval(id):** Cancela el intervalo el intervalo de tiempo referenciado por 'id', establecido por el método setInterval().

**clearTimeout(nombre):** Cancela el intervalo el intervalo de tiempo referenciado por 'nombre', establecido por el método setTimeout().

**confirm(mensaje):** Muestra el 'mensaje' en un cuadro de diálogo 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 y dispara el evento onFocus.

**prompt(mensaje,respuesta):** Muestra el 'mensaje' en un cuadro de diálogo que contiene una caja de texto con dos botones (*Aceptar* y *Cancelar*) en la cual podremos escribir una respuesta a lo que nos pregunte el 'mensaje'. El parámetro 'respuesta' es opcional, e inicialmente aparece en el campo de texto. Devuelve una cadena de caracteres con la respuesta introducida.

**setInterval(expresion,tiempo):** Evalúa la expresión cada vez que transcurren los milisegundos indicados en el segundo parámetro. Devuelve un valor que puede ser usado como identificativo por clearInterval().

**setTimeout(expresion,tiempo):** Evalúa la expresión después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearTimeout().

Ejemplo: Este ejemplo crea una barra de texto movible en la ventana del navegador. En este ejemplo se utilizan métodos del objeto String que trabaja con cadenas de caracteres, substring que obtiene parte de una cadena y length que obtiene la longitud de una cadena. Se verán más adelante. Se ha utilizado el evento onLoad= que se dispara inmediatamente después de que todos los objetos del documento se han transferido al navegador.

```
<HTML>
<HEAD>
<TITLE>Scroller en la barra de texto</TITLE>
<SCRIPT LANGUAGE="JavaScript" >
Texto = "Bienvenidos a mi página Web.....";
i = 0;
//Creamos el formulario
                document.write ('<FORM NAME="form1">');
    document.write ('<CENTER><INPUT NAME="text" VALUE= """"></CENTER>');
    document.write ('</FORM>');
//Escribimos la cadena
function MensajeScroll() {
    cadena=Texto.substring(i,Texto.length) + Texto.substring(0,i-1);
                if (i < Texto.length)   i++; else i = 0;
            document.form1.text.value=cadena

setTimeout("MensajeScroll()",50);
}
</SCRIPT>
</HEAD>
<BODY onLoad="MensajeScroll()">
</BODY>
</HTML>
```

***Ejemplo 31 – Scroller en la barra de desplazamiento***

**Ejemplo:** Vamos a añadir al ejemplo anterior un botón de forma que al hacer clic en él se imprima el contenido del documento que está visible en la ventana. Añadimos la función Imprimir con el método print():

```
function Imprimir()
{
window.print();
}
```

Y el código asociado al botón:

```
document.write ('<input type="button" value="Imprimir" name="imprimir"
onClick="Imprimir()">')
```

**Ejemplo:** Seguimos con el ejemplo anterior, ahora antes de imprimir nos ha de aparecer un cuadro de diálogo pidiéndonos confirmación para imprimir. Usaremos el método ***confirm()***. Ahora la función ***Imprimir()*** nos quedaría:

```
function Imprimir() {
if (confirm("¿Tiene encendida la impresora?"))
window.print();
}
```

Si el usuario pulsa el botón Aceptar, ***confirm()*** devuelve true y se ejecuta la sentencia para imprimir.

**Ejemplo:** El siguiente ejemplo define varios hipervínculos de forma que al pasar el puntero del ratón por cada hipervínculo se visualiza un mensaje en la etiqueta, al abandonarlo se dejará la etiqueta en blanco. El código para los hipervínculos es:

```
<form id="form1">
<p><b>Para visitar el Ejemplo 23 haz clic
<a href="Ejemplo23.htm" onMouseover="label1.value='Confirmación para
imprimir';return true;" onMouseout="label1.value='';return true;">aquí</a>
, Para ver el ejemplo 24 haz clic
<a href="Ejemplo 24.htm" onMouseover="label1.value='Scroller e impresión';return true;"
onMouseout="label1.value='';return true;">aquí </a><br>
<label for="label1">Mensaje:</label>
<input type="text" id="label1" name="fnombre" size="40" />
</b></p>
```

El evento **onMouseover** se dispara cuando el usuario pasa el puntero del ratón por el hipervínculo, onMouseout se dispara al salir del hipervínculo. Si la sentencia return true no aparece, el mensaje no se visualiza en la línea de estado.

**Ejemplo:** El siguiente ejemplo crea una ventana hija donde se carga el documento. En este documento aparecerán una serie de hipervínculos a ejemplos anteriores. Al hacer clic en estos hipervínculos se cargarán en la ventana padre inicial.

```
<HTML>
<HEAD>
<TITLE>OBJETOS - Ejemplo - Crea una ventana Hija</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT" >
ventanahija = window.open("Ejemplo 31.html", "miventana", "width=175,height=175")
</SCRIPT>
</HEAD>
<BODY >
<CENTER>
<B><P>ESTA PÁGINA CREA UNA VENTANA CON UN INDICE A DIVERSOS EJEMPLOS</P></B>
<P>LOS EJEMPLOS SE CARGARÁN EN ESTA VENTANA</P>
</CENTER>
</BODY>
</HTML>
```

***Ejemplo 32 – Ventanas hijas***

```
<HTML>
<HEAD>
<TITLE>OBJETOS - Ventana hija del Ejemplo anterior</TITLE>
<SCRIPT LANGUAGE="JAVASCRIPT" >
function ColocarUrl(direccion) {
opener.document.location = direccion;
}
</SCRIPT>
</HEAD>
<BODY >
<CENTER>
<p><b>HAZ CLIC EN UN EJEMPLO</p></b>
<A HREF="javascript:ColocarUrl('Ejemplo_23.htm')">Ejemplo 23</A><BR>
<A HREF="javascript:ColocarUrl('Ejemplo_24.htm')">Ejemplo 24</A><BR>
<A HREF="javascript:ColocarUrl('Ejemplo_25.htm')">Ejemplo 25</A><BR>
</CENTER>
</BODY>
</HTML>
```

***Ejemplo 33 – Ventana hija del ejemplo anterior***

***opener*** contiene la referencia a la ventana que abrió ésta, es el documento padre, con la sentencia ***opener.document.location=direccion*** se cargará en dicha ventana el documento indicado en dirección, también se podría haber puesto ***opener.window.location.***

## 5.6.5 -EL OBJETO LOCATION

Representa la información de la URL de cualquier ventana que esté actualmente abierta. Ya hemos visto en ejemplos anteriores cómo puede obtenerse la URL de un documento; y de un documento contenido en frames. Las propiedades y métodos se resumen en esta tabla:

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **Hash, host, hostname, href, pathname, port, protocol, search** | **reload(), replace(cadenaURL)** | **ninguno** |

Para asignar a la ventana actual una nueva dirección: [***window***]***.location="URL"*** Para acceder a las propiedades o métodos: [***window.***]***location.propiedad*** [***window.***]***location.método()*** [window] es opcional. Supongamos que tenemos la siguiente dirección: http://www.servidor.com:80/camino/servicios.htm#primero veamos cuales son las propiedades para la misma:

## 5.6.6 -PROPIEDADES DEL OBJETO LOCATION

**hash:** Cadena que contiene el nombre del enlace, dentro de la URL. "#primero".

**host:** Contiene el nombre del servidor y el número del puerto.: "www.servidor.com:80"

**hostname:** Contiene el nombre de dominio del servidor (o la dirección IP): "www.servidor.com"

**href:** Contiene la URL completa:

"http://www.servidor.com:80/camino/servicios.htm#primero"

**pathname:** Contiene el camino al recurso: "/camino/servicios.htm"

**port:** Cadena que contiene el número de puerto del servidor: "80"

**protocol:** Contiene el protocolo utilizado (incluyendo los dos puntos): ":80"

**search:** Mantiene la parte de URL que contiene la información que se pasa en una llamada a un script CGI.

### 5.6.7 -MÉTODOS DEL OBJETO LOCATION

**reload():** Vuelve a cargar el documento cuya URL se mantiene en la propiedad href.

**replace(cadenaURL):** Carga el documento cuya URL se pasa como parámetro y hace que el historial actual sólo contenga esa dirección, no se podrá volver a la página anterior usando el botón Atrás del navegador.

**Ejemplo**: Veamos un ejemplo de uso de estas propiedades.

```
<SCRIPT LANGUAGE="JAVASCRIPT" >
function localizar(){
document.write("HREF: " + location.href +"<br>"); document.write("HOST: " +
location.host +"<br>"); document.write("HOSTNAME: " + location.hostname +"<br>");
document.write("PATHNAME: " + location.pathname +"<br>");
document.write("PORT: " + location.port +"<br>");
document.write("PROTOCOL: " + location.protocol +"<br>");
}
</SCRIPT>
```

**Ejemplo:** el siguiente ejemplo utiliza el método ***replace(),*** al hacer clic en el botón se carga un documento, y no se puede volver a la página anterior usando el botón Atrás del navegador:

```
<HTML>
<HEAD>
<TITLE>Objeto Location</TITLE>
</HEAD>
<BODY >
<form >
<p><input type="button" value="Uso de replace"
onclick="location.replace('Ejemplo_23.html')"></p>
</form>
</BODY>
</HTML>
```

***Ejemplo 34 – Objeto Location***

## 5.7 - EL OBJETO HISTORY.

Cuando navegamos por la red, el explorador guarda una lista de las últimas URL donde se ha estado. Este objeto se encarga de representar esta lista. Se utiliza, sobre todo, para movernos hacia delante o hacia atrás en dicha lista. Las propiedades y métodos se resumen en este cuadro:

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **length** | **back() forward() go(url)** | **ninguno** |

Para acceder a las propiedades o métodos: [window.]history.propiedad [window.]history.método() [window] es opcional

### 5.7.1 -PROPIEDADES DEL OBJETO HISTORY

**Length:** Contiene el número de entradas de la lista del historial.

### 5.7.2 -MÉTODOS DEL OBJETO HISTORY

**back():** Carga la URL de la entrada anterior del historial.

**forward():** Carga la URL de la entrada siguiente.

**go(url):** Carga la URL del documento especificado por 'url' del historial. La 'url' puede ser un entero, en cuyo caso indica la posición relativa del documento dentro del historial (si es mayor que 0 va hacia delante, si es menor que 0 hacia detrás y si es 0 vuelve a cargar la ventana); o puede ser una cadena de caracteres, en cuyo caso hace referencia a toda o parte de una URL que esté en el historial.

**Ejemplo:** el siguiente código define dos botones para ir a la página anterior y a la siguiente :

```
<form >
<p><input type="button" value="Atras"
onclick="history.back()"></p>
<p><input type="button" value="Siguiente"
onclick="history.forward()"></p>
</form>
```

## 5.8 - EL OBJETO NAVIGATOR.

Este objeto no mantiene una relación directa con el resto de objetos del modelo de JavaScript, simplemente nos da información relativa al navegador que se esté utilizando para visualizar los documentos. Las propiedades y métodos se resumen en esta tabla.

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **appCodeName** | javaEnabled() | **ninguno** |
| **appName** | | |
| **appVersion** | | |
| **language** | | |
| **mimeTypes** | | |
| **platform** | | |
| **plugins** | | |
| **userAgent** | | |

Para acceder a las propiedades o métodos: ***navigator.propiedad navigator.método()***

### 5.8.1 -PROPIEDADES DEL OBJETO NAVIGATOR

**appCodeName:** Cadena que contiene el nombre del código del navegador del cliente.

**appName:** Contiene el nombre del navegador del cliente.

**appVersion:** Contiene la versión del navegador con el formato numerodeversión(plataforma;país)

**language:** Contiene información sobre el idioma de la versión del navegador.

**mimeTypes:** Array que contiene todos los tipos MIME soportados por el navegador del cliente.

**Platform:** Contiene el tipo de máquina en la que se compiló el navegador.

**Plugins:** Array que contiene todos los plug-ins soportados por el cliente.

**userAgent:** Contiene la cabecera del agente que se envía del cliente al servidor con el protocolo HTTP. Contiene la información de las propiedades appCodeName y appVersion.

### 5.8.2 -MÉTODOS DEL OBJETO NAVIGATOR

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

**Ejemplo:** El siguiente código javascript muestra las propiedades del objeto navigator.

```
<SCRIPT LANGUAJE="JavaScript">
var i,navi;
var navi=eval("navigator");
//eval evalúa el parámetro y devuelve el
// objeto asociado a la cadena
for (i in navi)
{
document.write("navigator. <b>" +i + "</b>=> <i>" + navi[i]+ "</i><br>")
}
</SCRIPT>
```

## 5.9 - PROPIEDADES Y MÉTODOS DE LOS OBJETOS DEL DOCUMENTO.

Fijémonos en el gráfico que muestra la jerarquía de objetos de JavaScript en el epígrafe 5.2.2, vemos que el objeto document abarca gran parte del modelo de

objetos. La mayor parte de comunicación entre el script y el usuario se realiza a través de este objeto y sus elementos, por ello es necesario comprender el campo de acción de dicho objeto.

### 5.9.1 -EL OBJETO DOCUMENT.

El objeto window y el objeto frame tienen asociado un objeto document que es el que realmente contiene el resto de los objetos como son: texto, imágenes, enlaces, formularios, etc que se muestran en el navegador. En la siguiente tabla se muestran las propiedades y métodos más significativos

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **alinkColor, anchors, applets, bgColor, cookie, fgColor , forms images, lastModified linkColor, links location , referrer title ,vlinkColor** | clear()<br>open()<br>close()<br>write()<br>writeln() | |

Para acceder a las propiedades y métodos del documento: [window.]document.propiedad [window.]document.método() [window] es opcional

### 5.9.2 -PROPIEDADES DEL OBJETO DOCUMENT:

**alinkColor:** Almacena el color de los enlaces activos.

**Anchors:** Array que contiene una entrada por cada enlace interno (etiqueta <A> con el atributo NAME) existente en el documento.

**Applets:** Array con los applets existentes en el documento.

**bgColor:** Almacena el color de fondo del documento.

**Cookie:** Cadena que contiene el valor de una cookie.

**fgColor:** Almacena el color del primer plano.

**Forms:** Array que contiene una entrada por cada formulario definido en el documento.

**Images:** Array con todas las imágenes del documento.

**lastModified:** Cadena con la fecha de la última modificación del documento.

**linkColor:** Color de los enlaces del documento.

**Links:** Array que contiene una entrada por cada enlace externo existente en el documento (etiquetas <AREA REF.=" "...> y <A REF.=" ">).

**Location:** Cadena con la URL del documento actual.

**Referrer:** Cadena con la URL del documento que llamó al actual, en caso de usar un enlace.

**Title:** Título del documento actual.

**vlinkColor:** Almacena el color de los enlaces visitados

### 5.9.3 -MÉTODOS DEL OBJETO DOCUMENT:

**open(mime,"replace"):** Abre la escritura sobre el documento. 'mime' es el tipo de documento soportado por el navegador. Si se indica "replace", se reutiliza el documento anterior (no crea una nueva entrada en el historial).

**close():** Cierra la escritura sobre el documento actual y fuerza la visualización de su contenido.

**write():** Escribe texto en el documento.

**writeln():** Escribe texto en el documento y lo finaliza con un salto de línea.

**Ejemplo:** El siguiente ejemplo muestra las características del documento: número de enlaces, de formularios, de imágenes, color de fondo, fecha última modificación, etc. En él se definen tres enlaces, uno a otro documento HTML y los otros dos a un ancla o enlace dentro del documento, dos imágenes y dos formularios.

```
<hr>
<A NAME="caracteristicas"></A>
<b><center>CARACTERISTICAS DEL DOCUMENTO</b><br>
<SCRIPT LANGUAJE="JavaScript">
document.write("Número de imágenes: "+ document.images.length+" <br>");
document.write("Número de enlaces externos: "+ document.links.length +" <br>");
document.write("Número de enlaces internos: "+ document.anchors.length+" <br>");
document.write("Nombre enlaces internos: "+ document.anchors[0].name +" , "
+ document.anchors[1].name + " <br>"); document.write("Número de formularios: "+
document.forms.length+" <br>"); document.write("Color de fondo: "+
document.bgColor+" <br>");
document.write("Color de los enlaces: "+ document.linkColor+" <br>");
document.write("Color de los enlaces activos: "+ document.alinkColor+" <br>");
document.write("Fecha última modificación: "+ document.lastModified+" <br>");
</SCRIPT>
</center>
```

**Ejemplo:** El siguiente ejemplo creará un documento HTML según unas características definidas de colores, al pulsar un botón se eligen unos colores, al pulsar otro botón se eligen otros colores; este nuevo documento se creará en un script. Se utilizará un array de dos elementos coloresdocu que contiene los textos que se visualizarán en el área definida en el formulario, informando sobre los colores con los que se creará y visualizará el nuevo documento. La función mostrar() visualiza en esta área el contenido del array, dependiendo del botón pulsado. La función pintapagina() crea el documento HTML, con las especificaciones de colores según hayamos pulsado un botón u otro. Invoca a la función Colores() o SinColores(). Las funciones Colores() y SinColores() devuelven una cadena que contiene los colores de fondo, de texto, de link y de link visitados.

```
<html>
<head>
<title>Objeto Document</title>
<SCRIPT LANGUAGE="JAVASCRIPT">
coloresdocu = new Array ();
//array con los textos que se visualizan en el area
 coloresdocu[0] = "Fondo: Blanco  , Color Texto: Negro, Link: Azul   ,  Link visitados: Rojo";
 coloresdocu[1] = "Fondo: Amarillo, Color Texto: Azul , Link: Rojo   ,  Link visitados: Rosa";

 var ventana=window.open("","","height=150, width=300") //ventana que se abre

 function  pintapagina(x){ //esta función escribe el texto del nuevo docum.
  var texto="";
  texto ="<HTML><HEAD><TITLE>PAGINA QUE CAMBIA DE COLORES</TITLE></HEAD><BODY ";
  if (x==1)
    texto=texto + Colores();
  else
    texto=texto + SinColores();
  texto=texto + "> <p>ESTA ES MI PAGINA WEB </p>";
  texto=texto + "<p><a href='Ejemplo_23.html'> Visita otro ejemplo</a></p>"
  texto=texto + "</BODY> </HTML>"
  ventana.document.open(); //abre la escritura sobre el documento
  ventana.document.write(texto);//escribe en el documento
          ventana.document.focus();  // muestra la ventana pequeña en primer plano
  ventana.document.close();  //cierra la escritura del documento
 }
 function SinColores(){  //devuelve las carac. Para el doc sin colores
   return "BGCOLOR='#ffffff' VLINK='#ff0000' LINK='#0000ff' TEXT='#000000'";
 }
 function Colores(){  //devuelve las carac. Para el doc con colores
  return "BGCOLOR='#ffff00' VLINK='#ff00ff' LINK='#ff0000' TEXT='#0000ff'";
 }
 function mostrar(i) {          //muestra el texto en el area
          document.FORMULARIO.area.value = coloresdocu[i]
 }
</SCRIPT></head>
<body><p align="center"><b>CREA UN NUEVO DOCUMENTO HTML</b></p>
<center> <table border="0" cellpadding="0" cellspacing="0" height="108">
<tr> <td height="87" align="center">
<form name="FORMULARIO" >
<input type="button" value="Documento con colores" name="B1"
onclick="mostrar(1);pintapagina(1);">
<input type="button" value="Valores por defecto" name="B2"
onclick="mostrar(0);pintapagina(0);"><br>
<textarea rows="4" name="area" cols="20" READONLY WRAP> Haz clic en uno de los
botones</textarea>
</form>
</td></tr></table></center>
</bodv></html>
```

***Ejemplo 35 – Objeto Document***

## 5.10 - LOS OBJETOS LINK Y ANCHOR.

Si el documento contiene algún hiperenlace local o externo, el objeto document tendrá asociados distintos objetos link y anchor. El objeto link engloba todas las propiedades que tienen los enlaces externos al documento actual y el objeto anchor todas las propiedades que tienen los enlaces locales al documento actual. Las propiedades del objeto link se resumen en la siguiente tabla:

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **links[indice].target** | **ninguno** | **onClick** |
| **length** | | **onMouseOver** |
| **propiedades del objeto** | | |
| **location** | | |

Para acceder a las propiedades: [window.]document.links[índice].propiedad

### 5.10.1 - PROPIEDADES:

**Target:** Es una cadena que contiene el nombre de la ventana o del frame especificado en el parámetro TARGET. Para hacer referencia a cada uno de estos objetos y poder obtener el valor de sus propiedades usamos la expresión: document.links[i].propiedad Donde i es el enlace que se está explorando (0 es el primer enlace del documento, 1 el segundo, ...).

**Length:** Para obtener el número de enlaces externos: document.links.length Para obtener el número de enlaces internos: document.anchors.length

## 5.11 - EL OBJETO IMAGE.

Este objeto nos permitirá manipular las imágenes del documento, pudiendo conseguir efectos como el conocido rollover, o cambio de imágenes, por ejemplo, al pasar el ratón sobre la imagen. Las propiedades se resumen en la siguiente tabla:

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **border, complete height, hspace, lowsrc, name, src, vspace, width** | | |

Para acceder a las propiedades: [window.]document.nombreimagen.propiedad

### 5.11.1 - PROPIEDADES DEL OBJETO IMAGE:

**Border:** Contiene el valor del atributo 'border' de la imagen.

**Complete:** Valor booleano que indica si la imagen se ha descargado completamente o no.

**Height:** Contiene el valor del atributo 'height' de la imagen (alto). Alto de la imagen que puede ponerse en píxeles o porcentaje con respecto a la página.

**Hspace:** Contiene el valor del atributo 'hspace' de la imagen (espacio horizontal alrededor de la imagen).

**Lowsrc:** Contiene el valor del atributo 'lowsrc' de la imagen. Con este atributo podemos indicar un archivo de la imagen de baja resolución.

**Name:** Contiene el valor del atributo 'name' de la imagen. Se corresponde con el nombre qye se le da al objeto imagen.

**Src:** Contiene el valor del atributo 'src' de la imagen <IMG SRC=...>.

**Vspace:** Contiene el valor del atributo 'vspace' de la imagen. Sirve para indicar el espacio libre en vertical medido en píxeles que se coloca entre la imagen y los otros elementos que la rodean.

**Width:** Contiene el valor del atributo 'width' de la imagen. Ancho de la imagen que puede ponerse en píxeles o porcentaje con respecto a la página.

**Ejemplo:** El siguiente ejemplo muestra como crear imágenes de sustitución. En el documento aparecen dos imágenes (de nombres img1 e img2) que son dos hipervínculos, al pasar el ratón sobre ellas (evento onMouseover) la imagen cambia, cuando el ratón abandona la imagen (evento onMouseout) esta vuelve a cambiar. El código es el siguiente:

```
<HTML>
<HEAD>
<TITLE>Manipulando imágenes</TITLE>
</HEAD>
<BODY bgcolor="#FFFFCC" >
<center>
<A HREF="departa.htm" onMouseover="document.img1.src='img1off.gif'"
onMouseout="document.img1.src='img1on.gif'">
<IMG SRC="img1on.gif" NAME="img1" ALT="Departamentos"></A>

<A HREF="equipo.htm" onMouseover="document.img2.src='img2off.gif'"
onMouseout="document.img2.src='img2on.gif'">
<IMG SRC="img2on.gif" NAME="img2" ALT="Equipo Directivo"></A>
</center>
</BODY>
</HTML>
```

***Ejemplo 36 – Manipulando imágenes***

Para que cambie la imagen al pasar el ratón por encima o al abandonarla cambiamos el valor de la propiedad src de dicha imagen:

***document.nombreimagen.src='nuevaimagen'*** También se podía haber puesto:

***document['nombredeimagen'].src='nuevaimagen'*** Es decir:

***document.img2.src='img2off.gif' ó document['img2'].src='img2off.gif'***

**Ejemplo:** Este ejemplo obtiene la misma salida que el ejemplo anterior pero se ha añadido código javascript para cargar las imágenes en la memoria caché del navegador y colocarlas en variables:

```
<HTML>
<HEAD>
<TITLE>OBJETOS - Ejemplo IMAGENES de sustitución con función</TITLE>
<SCRIPT language=JavaScript>
//comprobar si el navegador entiende objetos imagen
 if (document.images) {
img1on=new Image();   img1on.src= "img1on.gif";
img2on=new Image();   img2on.src= "img2on.gif";
img1off=new Image();   img1off.src= "img1off.gif";
img2off=new Image();   img2off.src= "img2off.gif";
}
function imagenOn(NombreImagen) {
if (document.images) {
document[NombreImagen].src=eval(NombreImagen + "on.src");
}
}
function imagenOff(NombreImagen) {
if (document.images) {
document[NombreImagen].src=eval(NombreImagen + "off.src");
}
}
</SCRIPT>
</HEAD>
<BODY bgcolor="#FFFFCC" >
<center>
<A HREF="departa.htm" onMouseover="imagenOff('img1')" onMouseout="imagenOn('img1')" >
<IMG SRC="img1on.gif" NAME="img1" ALT="Departamentos"></A>
<A HREF="equipo.htm" onMouseover= "imagenOff('img2')" onMouseout="imagenOn('img2')">
<IMG SRC="img2on.gif" NAME="img2" ALT="Equipo Directivo"></A>
</center>
</BODY>
</HTML>
```

***Ejemplo 37 – Manipulando imágenes con eventos***

La sentencia *if (document.images)* comprueba si el navegador entiende los objetos imagen. Si es así se crean 4 objetos imagen y se les asigna el fichero de imagen correspondiente usando la propiedad src:

*img1on=new Image();  img1on.src= "img1on.gif";*

Se han creado dos funciones para el cambio de las imágenes: ***imagenOn(NombreImagen)***que realiza el intercambio para los nombres de imagen que terminan en on e ***imagenOff(NombreImagen)*** que lo realiza para las imágenes que terminan en off. Ambas realizan la siguiente operación:

***document[NombreImagen].src=eval(NombreImagen + "on.src");***

NombreImagen es el nombre dado a la imagen (NAME='img1', NAME ='img2'). Asigna a ***document[NombreImagen].src*** el resultado de evaluar la expresión que concatena NombreImagen con "on.src" o con "off.src" dependiendo si se quiere visualizar la imagen que termina en on o en off, el resultado es img1on.src ó img1off.src, etc

Estas dos funciones se pueden simplificar en una de la siguiente forma:

```
function CambiarImagen(NombreImagen, NuevaImagen)
{
if (document.images)
{
document[NombreImagen].src=eval(NuevaImagen + ".src");
}
}
```

La llamada sería: ***CambiarImagen('img1','img1on')***, img1 es el nombre de imagen del objeto e img1on es una de las variables imagen creadas a las que luego se les asigna el archivo de imagen.

**Ejemplo:** Partimos del ejemplo anterior, vamos a crear dos hipervínculos de forma que ahora al pasar el ratón por el hipervínculo hacemos que cambie la imagen. El código Javascript será el mismo que el usado en el ejemplo anterior.

```
<BODY bgcolor="#FFFFCC" >
<center>
<IMG SRC="img1on.gif" NAME="img1" ALT="Departamentos">
<IMG SRC="img2on.gif" NAME="img2" ALT="Equipo Directivo">
<BR>
<A HREF="departa.htm" onMouseover="imagenOff('img1')"
onMouseout="imagenOn('img1')" >Departamentos</A> 
<A HREF="equipo.htm" onMouseover= "imagenOff('img2')"
onMouseout="imagenOn('img2')">Equipo Directivo </A>
</center>
</BODY>
</HTML>
```

**Ejemplo**: El siguiente ejemplo muestra las características de la imagen que aparece en el documento, como el nombre, el alto, el ancho, el borde, etc. Se ha añadido el siguiente código JavaScript para obtener dichas características:

```
<p align="center"><img border="5" src="glaciar.gif" alt="LAGUNA GLACIAR"
hspace="15" vspace="5" width="342" height="211"  NAME="laguna"> </p>
<center>
<br><b>CARACTERÍSTICAS DE LA IMAGEN:</b><br>
<SCRIPT LANGUAJE="JavaScript">
  document.write("Número de imágenes: "+ document.images.length+" <br>");
  document.write("Nombre de la imagen: "+ document.images[0].name+" <br>");
  document.write("BORDE: "+ document.laguna.border );
  document.write("  ALTO: "+ document.laguna.height);
  document.write("  ANCHO: "+ document.laguna.width+" <br>");
  document.write("Nombre de fichero: "+ document.laguna.src+" <br>");
  document.write("Espacio vertical alrededor de la imagen: "+
 document.laguna.vspace +" <br>");
  document.write("Espacio horizontal alrededor de la imagen: "+
 document.laguna.hspace+" <br>");
</SCRIPT>
<center>
```

## 5.12 - LOS OBJETOS DEL FORMULARIO.

En este apartado veremos cómo manipular los objetos de un formulario, así podremos hacer funciones que nos permitan validarlo antes de enviar los datos a un

servidor. JavaScript define un conjunto de objetos relacionados con los elementos que se pueden incluir dentro de un formulario, son los siguientes: form, text, textarea, button, checkbox, radio, select, password y hidden. Toda etiqueta <FORM> de HTML tiene asociada un objeto form de JavaScript.

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **action, elements, encoding length , method, target** | **reset() submit()** | **OnSubmit=** |

Para acceder a las propiedades o métodos:

***[window.]document.nombreformulario.propiedad,***

***[window.]document.nombreformulario.método()***

***[window.]document.forms[índice].propiedad***

***[window.]document.forms[índice].método()***

Si en el documento tenemos tres formularios podemos utilizar ***document.forms[0]*** para referenciar el primer formulario, ***document.forms[1]*** para el segundo y ***document.forms[2]*** para el tercero.

### 5.12.1 - PROPIEDADES DEL OBJETO FORM

**Action:** Cadena que contiene el valor del atributo ACTION del formulario (URL del programa que procesa los datos del formulario).

**Elements:** Array que contiene todos los elementos del formulario, en el mismo orden en el que se definen en el documento HTML. Por ejemplo, si en el formulario hemos puesto, en este orden, una caja de texto, un checkbox y una lista de selección, la caja de texto será elements[0], el checkbox será elements[1] y la lista de selección será elements[2].

**Encoding:** Cadena que contiene el valor del atributo ENCTYPE del formulario (código MIME).

**Length:** Número de elementos que contiene el formulario.

**Method:** Cadena que contiene el valor del atributo METHOD del formulario (método con el que se va a recibir/procesar la información del formulario GET/POST).

**Target:** Cadena que contiene el valor del atributo TARGET (nombre de ventana o marco donde aparecerá la respuesta que genere el servidor después de enviar el formulario).

### 5.12.2 - MÉTODOS DEL OBJETO FORM

reset(): Simula el mismo efecto que si pulsáramos un botón de tipo RESET dispuesto en el formulario (borra los datos que haya introducido el usuario en el formulario).

submit(): Envía el formulario. Tiene el mismo efecto que si pulsáramos un botón de tipo SUBMIT dispuesto en el formulario. Recordemos la sintaxis para crear un formulario:

```
<FORM NAME="NombreFormulario" [TARGET="NombreVentana"] [ACTION="URLservidor"]
[METHOD= GET | POST] [ENCTYPE="tipoMIME"] [onSubmit="Función_texto"] >
...objetos del formulario...
</FORM>
```

Si el formulario no va a tener peticiones o envíos a través del servidor los atributos ACTION=, TARGET= y METHOD= no son necesarios. Pero si el formulario va a realizar peticiones o preguntas al servidor, será necesario especificar al menos los atributos ACTION= y METHOD=; especificaremos el atributo TARGET= si los datos resultantes del servidor se van a mostrar en otra ventana que no sea la que realiza la llamada. Igualmente especificaremos el atributo ENCTYPE= si los script de formulario dan forma a los datos ligados al servidor en un tipo MIME= que no sea un flujo ASCII.

En este apartado aprenderemos a utilizar JavaScript para validar los datos del formulario, es decir para asegurarnos que los formularios contienen información válida antes de que se envíen al servidor.

**Ejemplo:** El siguiente ejemplo muestra algunas de las propiedades del formulario.

```
<SCRIPT LANGUAGE="JAVASCRIPT" >
function localizar(){
document.write("HREF: " + location.href +"<br>"); document.write("HOST: " +
location.host +"<br>"); document.write("HOSTNAME: " + location.hostname +"<br>");
document.write("PATHNAME: " + location.pathname +"<br>");
document.write("PORT: " + location.port +"<br>");
document.write("PROTOCOL: " + location.protocol +"<br>");
}
</SCRIPT>
```

Para obtener el número de elementos del formulario también podríamos haber utilizado las siguientes expresiones: **document.forms[0].length**, o bien, **document.FORMULARIO.length**

Donde FORMULARIO es el nombre dado al formulario y forms (propiedad del objeto document) es un array que contiene una entrada por cada formulario definido en el documento (forms[0] es el primer formulario que se encuentra en el documento).

**Ejemplo**: Partimos del ejemplo anterior, en este ejemplo añadimos a uno de los botones el evento onClick, de forma que al hacer clic en el botón comprobaremos si se ha dejado algún campo en blanco, en ese caso aparecerá un cuadro de diálogo indicándolo, y el foco situará el puntero del ratón en el campo . El código añadido al botón es: **<input type="button" value="Enviar" name="enviar" onClick="Comprobar()">**

```
<<SCRIPT language="JavaScript">
<!--
function Comprobar(){
var i;
var n;
n=document.forms[0].elements.length; //nº de elementos
for ( i=0; i<n ; i++)  //recorremos elementos del formulario
{
if(document.forms[0].elements[i].value==""){
alert("Has de introducir datos en el campo: "+
document.forms[0].elements[i].name);
document.forms[0].elements[i].focus();
break   //rompe el bucle
}
}
}
// -->
</SCRIPT>
```

Para obtener el valor de un elemento *i* del formulario utilizamos la expresión: ***document.forms[0].elements[i].value***. Para obtener el nombre de un elemento del formulario utilizamos la expresión: ***document.forms[0].elements[i].name***. Para que el foco de ejecución se sitúe en un campo utilizamos el método ***focus()*** que se verá en el siguiente epígrafe: ***document.forms[0].elements[i].focus();***

Casi todas las etiquetas HTML utilizadas para definir elementos en un formulario tienen asociado un objeto JavaScript.

## 5.13 - LOS OBJETOS TEXT, TEXTAREA Y PASSWORD.

Estos objetos son el medio principal para la obtención de texto introducido por el usuario. Representan los campos de texto y las áreas de texto dentro de un formulario. El objeto password es igual que el objeto text salvo que los caracteres introducidos por el usuario se muestran poniendo asteriscos (*) en su lugar. El objeto textarea está destinado  para la introducción de múltiples líneas de texto. Los tres tienen las mismas propiedades y métodos, por ello los vemos juntos.

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **defaultValue**<br>**name**<br>**value** | **blur()**<br>**focus()**<br>**select()** | **onBlur**<br>**onFocus**<br>**onChange**<br>**onSelect** |

Para acceder a las propiedades o métodos:

***[window.]document.nombreformulario.nombrecampo.propiedad,***

***[window.]document.nombreformulario.nombrecampo.método()***

***[window.]document.nombreformulario.elements[n].propiedad***

***[window.]document.nombreformulario.elements[n].método()***

***[window.]document.forms[m].nombrecampo.propiedad***

***[window.]document.forms[m].nombrecampo.método()***

***[window.]document.forms[m].elements[n].propiedad***

***[window.]document.forms[m].elements[n].método()***

### 5.13.1 - PROPIEDADES DE LOS OBJETOS TEXT, TEXTAREA Y PASSWORD

**defaultValue:** Cadena que contiene el valor por defecto dado al objeto (valor del parámetro VALUE).

**Name:** Cadena que contiene nombre del campo (valor del parámetro NAME).

**Value:** Cadena que contiene el valor actual del campo.

### 5.13.2 - MÉTODOS DE LOS OBJETOS TEXT, TEXTAREA Y PASSWORD

**blur():** Elimina el foco del objeto.

**focus():** Asigna el foco del ratón al objeto.

**select():** Selecciona el texto que contiene el objeto

**Ejemplo:** En el siguiente formulario pediremos al usuario su nombre y la contraseña y de nuevo la contraseña para validarla. Si las contraseñas no coinciden se visualizará un cuadro de diálogo indicándolo, entonces el foco de la ejecución volverá al campo contraseña (método focus()) que además aparecerá seleccionado (método select()). Si no se teclea nada en el campo contraseña también aparecerá un mensaje y el foco retornará a dicho campo. Si los datos han sido validados se enviarán al servidor (método submit()) a un programa CGI. El código es el siguiente:

```
<HTML>
<HEAD>
<TITLE> Comprobar Password</TITLE>
<SCRIPT language="JAVASCRIPT">
function validarFormulario(formulario) {
        if (formulario.clave1.value == "") {
          //no se ha introducido contraseña
                alert("Debes introducir la contraseña");
                formulario.clave1.focus(); //devolver el foco al campo
                return;  }
        if (formulario.clave1.value != formulario.clave2.value) {
          //las contraseñas escritas no coinciden
                alert("Las contraseñas introducidas son distintas")
                formulario.clave1.focus(); //devolver el foco
                formulario.clave1.select(); //seleccionar texto
                return;  }
    formulario.action="http://www.servidor.es/login_servidor.cgi"
    formulario.submit();  //enviar datos al servidor
}
</SCRIPT>
</HEAD>
<BODY>
<p align="center"><b>VERIFICACIÓN DE LA CONTRASEÑA</b>
<center><table bgcolor="#FFFFCC" border="2">
<td>
<FORM onSubmit="javascript:validarFormulario(this)" method="POST"> Escribe tu nombre:
<INPUT size=30 name="NOMBRE">
<P>Contraseña: <INPUT name="clave1" type=password size="20">
<P>Introduce de nuevo la contraseña: <INPUT name="clave2" type=password size="20">
<P><center>
<INPUT type="submit"  value=Enviar name="Enviar"> 
<INPUT type="reset" value=Restablecer>
</center>
</td></table></center></FORM>
</BODY></HTML>
```

***Ejemplo 38 – Objetos formulario: Comprobar password***

El evento onSubmit se dispara cuando un formulario está a punto de enviarse. Cuando se envíe el formulario (se hace clic en el botón Enviar, type="submit") se invoca a la función validarFormulario(), el parámetro que se envía es this que hace referencia al objeto actual, es decir al formulario. Se ha definido el método POST para enviar la información al servidor, method="POST". Cuando el formulario haya sido validado se realizan las siguientes acciones:

```
formulario.action="http://www.servidor.es/login_servidor.cgi"

formulario.submit();  //enviar datos al servidor
```

Es decir la acción a realizar será enviar los datos al servidor, a un programa CGI que se encargará de procesar la información; a la URL indicada por el atributo action. Este formulario es un ejemplo de cómo enviar datos al servidor para ser procesados por un archivo CGI. Más detalles sobre transmisión de datos a archivos CGI no es objetivo del curso.

## 5.14 - LOS OBJETOS BUTTON, RESET Y SUBMIT.

Definen los tres tipos de botones que se pueden incluir en un formulario. Puede ser un botón genérico, ***"button"***, que no tiene acción asignada, un botón ***"submit"*** que al ser pulsado envía los datos del formulario o un botón ***"reset"*** que al ser pulsado limpia los valores del formulario.

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **name**<br>**value** | **click()** | **onClick** |

### 5.14.1 - PROPIEDADES DE LOS OBJETOS BUTTON RESET Y SUBMIT

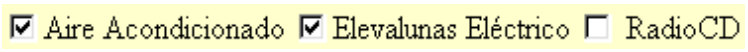
**Name:** Cadena que contiene el valor del parámetro NAME.

**Value:** Cadena que contiene el valor del parámetro VALUE.

### 5.14.2 - MÉTODOS DE LOS OBJETOS BUTTON

**click():** Reproduce la acción que el usuario realiza cuando hace clic en un botón.

## 5.15 - EL OBJETO CHECKBOX.

Las casillas de verificación o "checkboxes" nos permiten seleccionar varias opciones marcando el cuadrito que aparece a su izquierda 

. La marca de la casilla equivale a un valor “true” y si no está marcada equivale a un valor “false”.

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **Checked**<br>**defaultChecked**<br>**name**<br>**value** | **click()** | **onClick** |

### 5.15.1 - PROPIEDADES DEL OBJETO CHECKBOX:

**Checked:** Valor booleano que indica si la checkbox está seleccionada (true) o no seleccionada (false).

**defaultChecked:** Valor booleano que indica si la checkbox debe estar seleccionado por defecto o no.

**Name:** Cadena que contiene el valor del parámetro NAME.

**Value:** Cadena que contiene el valor del parámetro VALUE.

### 5.15.2 - MÉTODOS DE LOS OBJETOS CHECKBOX:

**click():** Reproduce la acción que el usuario realiza cuando hace clic en un botón.

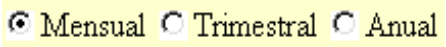
**Ejemplo:** En el siguiente formulario pediremos al usuario que introduzca su nombre y seleccione las casillas deseadas. Los botones de opción no tienen funcionalidad en este ejemplo. Al hacer clic en el botón Calcular, se visualizará en el campo TOTAL un total pts que dependerá de las casillas seleccionadas (Figura5_21).

```
<p><b>Accesorios</b><b>:
<input type="checkbox" name="AIRE" ></b>Aire Acondicionado
<input type="checkbox" name="ELEVALUNAS" >Elevalunas Eléctrico
<input type="checkbox" name="RADIO" > RadioCD
</p>
<CENTER>
<input type="button" value="Calcular" name="Calcular" onClick="Calculo()">
<b>TOTAL:</b> <input type="text" name="TOTAL" size="10" value=0
READONLY>Pts</CENTER>
La función Calculo() es:
<SCRIPT language="JavaScript">
<!--
function Calculo(){
var tot=0;
if (document.FORMULARIO.AIRE.checked)  tot=tot+120000;
if (document.FORMULARIO.ELEVALUNAS.checked) tot=tot+50000;
if (document.FORMULARIO.RADIO.checked) tot=tot+40000;
document.FORMULARIO.TOTAL.value=tot;
}
// -->
</SCRIPT>
```

Para comprobar si se ha seleccionado una casilla, por ejemplo la casilla AIRE, utilizamos la expresión: ***if (document.FORMULARIO.AIRE.checked)*** Devuelve true si la casilla está seleccionada, en caso contrario devuelve false.

## 5.16 - EL OBJETO RADIO

Este objeto nos permitirá elegir una posibilidad entre todas las que hay:




Todos los botones de un grupo van a compartir el mismo nombre, de esta manera JavaScript conoce al grupo de botones de tal forma que al hacer clic en uno de ellos se desactive el resto de botones del grupo.

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| **checked**<br>**defaultChecked, length**<br>**name, value** | **click()** | **onClick()** |

Para acceder a las propiedades o métodos:

***[window.]document.nombreformulario.grupobotones[x].propiedad***

***[window.]document.nombreformulario.grupobotones[x].método()***

***[window.]document.nombreformulario.elements[n].propiedad***

***[window.]document.nombreformulario.elements[n].método()***

***[window.]document.forms[m].grupobotones.propiedad***

***[window.]document.forms[m].grupobotones.método()***

***[window.]document.forms[m].elements[n].propiedad***

***[window.]document.forms[m].elements[n].método()***

### 5.16.1 - PROPIEDADES DEL OBJETO RADIO:

**checked:** Valor booleano que nos dice si el radio está seleccionado (true) o no (false).

**defaultChecked:** Valor booleano que indica el elemento radio debe estar seleccionado por defecto o no

**length:** Número de botones de opción definidos en el grupo de elementos radio.

**name:** Cadena que contiene el valor del parámetro NAME.

**value:** Cadena que contiene el valor del parámetro VALUE.

### 5.16.2 - MÉTODOS DEL OBJETO RADIO:

**click():** Reproduce la acción que el usuario realiza cuando hace clic en un botón.

**Ejemplo:** Partimos del formulario anterior, añadimos un nuevo botón que visualizará en una nueva ventana el valor del botón de radio seleccionado y las casillas seleccionadas. El código HTML para el nuevo botón es:

```
<input type="button" value="Ver selecciones" name="VER" onClick="VerSeleccion()">
```

```
<function VerSeleccion(){
var i;
var ventana;

ventana=window.open("","",'width= 340,height=150');
ventana.document.write("<B>BOTON SELECCIONADO:</B><BR>");

// bucle que recorre los botones
for (i=0; i<document.FORMULARIO.BOTONES.length; i++)
if(document.FORMULARIO.BOTONES[i].checked){ ventana.document.write("Valor del
botón => "+ document.FORMULARIO.BOTONES[i].value +"<BR>");
}

ventana.document.write("<B>CASILLAS SELECCIONADAS:</B><BR>");

for (i=4;i<7;i++) //casillas elementos 4 a 6
if(document.FORMULARIO.elements[i].checked){ ventana.document.write("Nombre de
casilla: "+document.FORMULARIO.elements[i].name );
ventana.document.write(" * Valor: "+document.FORMULARIO.elements[i].value +"<BR>");
}
}
```

Para saber el número de botones de opción del grupo de botones usamos la expresión: ***document.FORMULARIO.BOTONES.length***; por ello usamos un bucle hasta el valor anterior para comprobar si el botón está chequeado o no; que lo comprobamos con la expresión: ***if(document.FORMULARIO.BOTONES[i].checked)*** Para obtener el valor del botón de radio tenemos la expresión: ***document.FORMULARIO.BOTONES[i].value***

## 5.17 - EL OBJETO SELECT.

Este objeto representa una lista de opciones dentro de un formulario. Pueden aparecer en una página como cuadros de lista desplegables y por otra como cuadros de lista. Presentan un uso eficaz a la hora de presentar lista de opciones por tratarse de una lista desplegable de la que podremos escoger alguna (o algunas) de sus opciones.

| Propiedades | Métodos | Manejadores de eventos |
|---|---|---|
| length<br>name<br>options<br>selectedIndex<br>options[n].defaultSelected<br>options[n].index<br>options[n].selected<br>options[n].text<br>options[n].value | Ninguno | onChange() |

Para acceder a las propiedades:

*[window.]document.nombreformulario.nombrelista.propiedad*

*[window.]document.forms[m].nombrelista.propiedad*

*[window.]document.nombreformu.nombrelista.options[n].propiedad*

*[window.]document.forms[m].nombrelista.options[n].propiedad*

### 5.17.1 - PROPIEDADES DEL OBJETO SELECT:

**Length:** Número de opciones definidas en la lista.

**Name:** Contiene el valor del parámetro NAME.

**Options:** Se trata de un array que contiene una entrada por cada una de las opciones de la lista, en el mismo orden en el que aparecen en el código HTML. Este array tiene, a su vez, las siguientes propiedades:

**defaultSelected:** indica si la opción debe estar seleccionada por defecto (true, si; false no).

**index:** indica la posición de la opción dentro de la lista.

**selected:** indica si la opción está actualmente seleccionada o no (true, si; false no).

**text:** contiene el texto de la opción.

**value:** contiene el valor del parámetro VALUE de la opción (valor que se envía asociado al elemento Option cuando se manda el formulario al servidor).

**selectedIndex:** Contiene el valor del índice de la opción actualmente seleccionada.

En el siguiente ejemplo tenemos que elegir un menú de comida a partir de un conjunto de opciones. Para el primer plato se crea una lista desplegable cuya definición es la siguiente:

```
<p><b>ELIGE EL PRIMER PLATO: </b>
<select size="1" name="PRIMERPLATO">
<option value="" selected>Selecciona primer plato</option>
<option value="Garbanzos" >Garbanzos</option>
<option value="Lentejas">Lentejas</option>
<option value="Judías Verdes">Judías Verdes</option>
<option value="Paella">Paella</option>
</select></p>
```

Si no elegimos ningún elemento del menú se visualizará un aviso indicándolo. Para elegir el segundo plato se crea un cuadro de lista cuya definición es la siguiente:

```
<p> <b>ELIGE SEGUNDO PLATO :</b>
<select size="3" name="SEGUNDOPLATO" multiple>
<option>Entrecot a la Plancha</option>
<option>Pescado a la Plancha</option>
<option>Cochinillo Frito</option>
<option selected>Huevos con bacon y chorizo</option>
</select></p>
```

Para elegir el postre definimos un grupo de botones de radio:

```
<p><b>POSTRE: </b>
<input type="radio" value="FRUTA" checked
name="POSTRE">Fruta
<input type="radio" name="POSTRE" value="HELADO">Helado
<input type="radio" name="POSTRE" value="YOGUR">Yogur
<input type="radio" name="POSTRE" value="OTRO">Otro
```

El botón Ver Menú visualiza en una nueva ventana el menú elegido por el usuario. Al hacer clic en el botón se invoca a la función VerSeleccion() enviando el parámetro **this.form** que hace referencia al formulario actual. El código HTML asociado al botón es:

```
<input type="button" value="Ver Menú" name="VER" onClick="VerSeleccion(this.form)">
```

El código JavaScript de la función VerSeleccion() es el siguiente:

```
<<SCRIPT language="JavaScript">
<!--
function VerSeleccion(form){
var ventana;

ventana=window.open("","",'width= 340,height=150'); ventana.document.write("<B>HAS
ELEGIDO EL SIGUIENTE MENÚ:</B><BR>"); ventana.document.write("<B>Primer plato:
</B>");

primero = form.PRIMERPLATO.selectedIndex; //obtener selección primer plato
if (form.PRIMERPLATO.options[primero].value == "") {
alert("NO HAS ELEGIDO PRIMER PLATO") //si no elegimos nada
}
ventana.document.write(form.PRIMERPLATO.options[primero].value +"<BR>");
//visualizamos el valor de la opción elegida

ventana.document.write("<B>Segundo Plato :</B>");
segundo = form.SEGUNDOPLATO.selectedIndex; //obtener selección 2º plato
ventana.document.write(form.SEGUNDOPLATO.options[segundo].text +"<BR>");
//Visualizamos el texto de la opción elegida

ventana.document.write("<B>Postre :</B>"); for (i=0; i<form.POSTRE.length; i++)
if(form.POSTRE[i].checked)
ventana.document.write(form.POSTRE[i].value +"<BR>");

ventana.document.close(); //cerramos la escritura en el documento
}
// -->
</SCRIPT>
```

Para obtener el elemento seleccionado de cada una de las listas se han usado estas expresiones:

*primero=form.PRIMERPLATO.selectedIndex*

*segundo=form.SEGUNDOPLATO.selectedIndex*

Para obtener el valor (debe estar definido el parámetro VALUE en la definición de la lista) seleccionado de la primera lista usamos esta expresión:

*form.PRIMERPLATO.options[primero].value*

Para obtener el texto de la opción seleccionada de la segunda lista usamos esta expresión:

*form.SEGUNDOPLATO.options[segundo].text*

## 5.18 - EL OBJETO HIDDEN

Este es el objeto oculto del formulario, contiene cadenas de caracteres cuyos contenidos no son visibles para el usuario de la página Web. Es similar a un objeto text salvo que no tiene valor por defecto y que no se puede editar. Son útiles para las aplicaciones CGI que implican múltiples pantallas y se suele utilizar para conservar información de estado entre las páginas.

### 5.18.1 - PROPIEDADES DEL OBJETO HIDDEN:

**Name:** Cadena que contiene el valor del parámetro NAME.

**Value:** Cadena que contiene el valor del parámetro VALUE.

Para acceder a las propiedades:

*[window.]document.nombreformulario.nombrecampo.propiedad*
*[window.]document.nombreformulario.elements[n].propiedad*
*[window.]document.forms[m].nombrecampo.propiedad*
*[window.]document.forms[m].elements[n].propiedad*

## 5.19 - PROPIEDADES Y MÉTODOS DE LOS OBJETOS DEL LENGUAJE.

Los objetos vistos hasta el momento son los que forman parte de la jerarquía de objetos JavaScript. Si embargo JavaScript posee otro tipo de objetos propios del lenguaje que permiten manejar nuevas estructuras de datos y añadir utilidades al lenguaje. Consideraremos en este apartado los objetos String, Date y Math.

### 5.19.1 - EL OBJETO STRING.

Este objeto nos permite la manipulación de cadenas de texto. Cuando asignamos una cadena a una variable, JavaScript crea un objeto de tipo String que es el que nos permite hacer las manipulaciones. Para acceder a las propiedades y métodos: ObjetoString.Propiedad ObjetoString.Método()

### 5.19.2 - PROPIEDADES DEL OBJETO STRING:

**length:** Nos indica la longitud en caracteres de la cadena dada. Ej: "Hola Mundo".length devuelve 10

**prototype:** Nos permite añadir nuevas propiedades al objeto String.

### 5.19.3 - MÉTODOS DEL OBJETO STRING:

**anchor(nombre_del_ancla):** Crea un enlace (local) asignando al atributo NAME el valor de 'nombre_del_ancla'. Este nombre debe estar entre comillas " ". Se asigna como texto del enlace el que tenga el objeto String. Genera el código HTML: <A NAME=”nombre_del_ancla”>valor del objeto string</A>.

**big():** Devuelve el valor del objeto string con una fuente grande. Genera el código HTML: <BIG>valor del objeto string</BIG>.

**blink():** Devuelve el valor del objeto string con un efecto intermitente. Genera el código HTML: <BLINK>valor del objeto string</BLINK>.

**bold():** Devuelve el valor del objeto string en negrita. Genera el código HTML: <B>valor del objeto string</B>.

**charAt(indice):** Devuelve el carácter situado en la posición especificada por 'indice' (el primer carácter ocupa la posición 0). Ej: "Hola Mundo".charAt(0) Î devuelve H

**concat(cadena):** Concatena el valor del objeto string con el que se pasa como parámetro. C1.concat(C2) Devuelve la cadena C1 concatenada con la cadena C2. Es equivalente a C1+C2. Ej: "Hola".concat("Mundo") devuelve HolaMundo

**fixed():** Devuelve el valor del objeto string con una fuente con formato monoespacio. Genera el código HTML: <TT>valor del objeto string</TT>

**fontcolor(color):** Cambia el color con el que se muestra la cadena. La variable color debe ser especificada entre comillas: " ", o bien siguiendo el estilo de HTML. Genera el código HTML: <FONT COLOR=" ">valor del objeto string</FONT>

**fontsize(tamaño):** Cambia el tamaño con el que se muestra la cadena. Los tamaños válidos son de 1 (más pequeño) a 7 (más grande).

**indexOf(cadena_buscada, indice):** Devuelve el lugar de la cadena actual donde se encuentra la primera ocurrencia de 'cadena_buscada' a partir de la posición dada por 'indice'. Este último argumento es opcional y, si se omite, la búsqueda comienza por el primer carácter de la cadena. Si no lo encuentra devuelve –1. Ej: "Hola Mundo".indexOf("n") devuelve 7

**italics():** Devuelve la cadena en cursiva. Genera el código HTML: <I>valor del objeto string</I>

**lastIndexOf(cadena_buscada ,indice):** Devuelve el lugar donde se encuentra la última ocurrencia de 'cadena_buscada' dentro de la cadena actual, a partir de la posición dada por 'indice', y buscando hacia atrás. Este último argumento es opcional y, si se omite, la búsqueda comienza por el último carácter de la cadena.

**link(URL):** Crea un enlace donde el atributo HREF toma el valor del URL y se asigna como texto del enlace el que tenga el objeto string. Genera el código HTML: <A REF.="URL">valor del objeto string</A>.

**small():** Devuelve el valor de la cadena con una fuente pequeña. Genera el código HTML: <SMALL>valor del objeto string</SMALL>.

**split(carácter):** Divide una cadena en subcadenas creadas a partir de la original de la siguiente forma: 1º subcadena: desde el comienzo hasta el carácter especificado (o hasta el final si no lo encuentra). 2ª y sucesivas: a partir del carácter especificado hasta el la siguiente ocurrencia del mismo o hasta el final. El carácter no se incluye en las subcadenas. "Hola Mundo".split(" ") devuelve Hola,Mundo

**strike():** Devuelve el valor de la cadena de caracteres tachada. Genera el código HTML: <STRIKE>valor del objeto string</STRIKE>

**slice(inicio,fin):** Devuelve una cadena formada formada por los caracteres que se encuentra entre la posición inicio y fin-1.

**sub():** Devuelve el valor de la cadena con formato de subíndice. Genera el código HTML: <SUB>valor del objeto string</SUB>

**substr(N1, N2):** Por ejemplo si considero C.substr(N1, N2): Devuelve una subcadena a partir de la cadena C tomando N2 caracteres desde la posición N1. Si no se especifica N2. Devolverá desde la posición N1 hasta el final de la cadena. Ej: "Hola Mundo".substr(2,5) devuelve la cadena Mu

**substring(N1,N2):** C.substring(N1,N2): Devuelve también una cadena a partir de la cadena C, pero en este caso N1 y N2 indican las posiciones de comienzo y de final de la subcadena. Ej: "Hola Mundo".substring(2,6) devuelve la cadena M

**sup():** Devuelve el valor de la cadena con formato de superíndice. Genera el código HTML: <SUP>valor del objeto string</SUP>

**toLowerCase():** Devuelve el valor de la cadena en minúsculas. Ej: "Hola Mundo".toLowerCase() devuelve al cadena hola mundo

**toUpperCase():** Devuelve la cadena en mayúsculas. Ej: "Hola Mundo".toUpperCase() devuelve la cadena HOLA MUNDO

**Ejemplo:** Este ejemplo muestra propiedades y métodos del objeto String. La orden with informa a JavaScript que el siguiente grupo de órdenes, dentro de los símbolos de llave, serán referidas a un objeto en particular de forma que no es necesario escribir la palabra document delante de los métodos write.

```
With (objeto) { [instrucciones]
}
Así en el ejemplo anterior utilizamos:
with(document) {
write("La cadena es: "+ cad +"<BR>");
write(.....)
.....
}
```

```
<html>
<head>
<title>Objeto String</title>
<SCRIPT LANGUAGE="JavaScript">
<!--
function cadenas()  {
var cad = "Hola Mundo";

with(document) {
write("La cadena es: "+ cad +"<BR>");
write("Longitud de la cadena: "+ cad.length +"<BR>");
write("Metodo anchor: "+ cad.anchor("Ancla") +"<BR>");
write("Negrita- bold: "+ cad.bold() +"<BR>");
write("Quinto caracter es: "+ cad.charAt(5) +"<BR>");
write("Formato FIXED: "+ cad.fixed() +"<BR>");
write("De color rojo- fontcolor: "+cad.fontcolor("#FF0000")+"<BR>");
write("Tamaño 5- fontsize: "+ cad.fontsize(5) +"<BR>");
write("<BR>En cursiva-Italics: "+ cad.italics() +"<BR>");
write("La primera <b>o</b> esta, empezando a contar por detras,");
write(" en la posicion- lastindexOf: "+cad.lastIndexOf("o") +"<BR>");
write("Haciendola enlace- link: "+cad.link("Ejemplo5_29.htm")+"<BR>");
write("Tachada- strike: "+ cad.strike() +"<BR>");
write("Subindice- sub: "+ cad.sub() +"<BR>");
write("Superindice- sup: "+ cad.sup()+"<BR>");
}
}
//-->
</SCRIPT></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
cadenas();
//-->
</SCRIPT> </BODY> </HTML>
```

***Ejemplo 39 – Objeto String***

Ejemplo: El siguiente ejemplo muestra en el área el código HTML que se genera al aplicar algunos de los métodos del objeto String. Por ejemplo: la cadena escrita se convertirá a mayúscula o minúscula. En las casillas elegimos el tipo de fuente, tamaño, etc ... de la lista elegimos un color, que se lo aplicaremos a la cadena escrita. Estas sentencias generan código HTML Al hacer clic en el botón se visualizará en el área la cadena escrita en mayúscula o minúscula, en forma de URL (incluyendo <A REF.=...>) y según el formato elegido.

```
<html>
<head>
<title>Objeto  String . Métodos</title>
<SCRIPT language="JavaScript">
<!--
function Formateatexto(){
var texto,colortexto;

texto=document.forms[0].cadena.value;
//comprobamos si no se ha tecleado nada
if(document.forms[0].cadena.value==""){
alert("¡Has de introducir datos en la cadena!");
document.forms[0].cadena.focus();
return;
}
//mayuscula o minuscula
if(document.FORMULARIO.BOTONES[0].checked) texto=texto.toUpperCase();
if(document.FORMULARIO.BOTONES[1].checked) texto=texto.toLowerCase();

//casillas de seleccion
if(document.FORMULARIO.NEGRITA.checked) texto=texto.bold();
if(document.FORMULARIO.ITALICA.checked) texto=texto.italics();
if(document.FORMULARIO.PEQUE.checked) texto=texto.small();
if(document.FORMULARIO.GRANDE.checked) texto=texto.big();

//lista de colores
ct=document.FORMULARIO.COLOR.selectedIndex;//opcion seleccionada
colortexto=document.FORMULARIO.COLOR.options[ct].value;
if(ct>0) //si no es la primera opcion texto=texto.fontcolor(colortexto)

//añadimos el link
document.FORMULARIO.resul.value=texto.link("Ejemplo_30.html");
}
//-->
</SCRIPT>
</head>
```

```
<body bgcolor="#FFFF99">
<form name="FORMULARIO">
<p><b>Escribe una cadena :</b> <input type="text" name="cadena" size="20">
<input type="radio" value="MAYUS" checked name="BOTONES"><b>Mayúsculas</b>
<input type="radio" name="BOTONES" value="MINUS"><b>Minúsculas</b>
</p>
<p><b><input type="checkbox" name="NEGRITA" >Negrita
<input type="checkbox" name="ITALICA" >Itálica
<input type="checkbox" name="PEQUE" >Pequeña
<input type="checkbox" name="GRANDE" >Grande</b>
<select size="1" name="COLOR">
<option selected value="">Elige un color</option>
<option value="#ff0000">Rojo</option>
<option value="#0000ff">Azul</option>
<option value="#ffff00">Amarillo</option>
<option value="#00ff00">Verde</option>
</select>
<textarea rows="5" name="resul" cols="28" READONLY WRAP> Cadena
resultante:</textarea></p>
<p><center><input type="button" onclick="Formateatexto()"
value="Ver cadena  resultante en forma de URL" name="boton"> </center></p>
</form>
<HR>
</body>
</html>
```

***Ejemplo 40 – Objeto String: Métodos***

## 5.19.4 - EL OBJETO MATH.

Este objeto nos permite poder realizar cálculos en los scripts. Sus propiedades sólo pueden consultarse, son constantes matemáticas de uso frecuente. Las referencias a las propiedades devuelven los valores inherentes (como por ejemplo pi); las referencias a los métodos requieren un valor enviado como parámetro al método y devuelven el resultado del método realizando la operación con el parámetro. La forma de usar el objeto Math es la misma que la forma de usar cualquier objeto JavaScript: Math.propiedad, Math.método()

| Propiedad | Descripción |
|---|---|
| **Math.E** | Constante de Euler. Número “e”, (aproximadamente 2.718) |
| **Math.LN2** | Logaritmo neperiano de 2 (aproximadamente 0.693) |
| **Math.LN10** | Logaritmo neperiano de 10 (aproximadamente 2.302) |
| **Math.LOG2E** | Logaritmo en base 2 de “e” (aproximadamente 1.442) |
| **Math.LOG10E** | Logaritmo en base 10 de “e” (aproximadamente 0.434) |
| **Math.PI** | Constante PI (aproximadamente 3.14159) |
| **Math.SQRT1_2** | Raíz cuadrada de 0.5 (aproximadamente 0.707) |
| **Math.SQRT2** | Raíz cuadrada de 2 (aproximadamente 1.414) |

| Función | Método | Descripción y ejemplo |
|---|---|---|
| **Raiz cuadrada** | Math.sqrt(N) | Devuelve la raíz cuadrada del argumento. Ej: Math.sqrt(27) devuelve 5.196152422706632 |
| **Potencia** | Math.pow(N1, N2) | Devuelve N1 elevado a N2. Ej: Math.pow(5, 3) devuelve 125 |
| **Valor absoluto** | Math.abs(N) | Devuelve el valor absoluto de un número (es decir, el valor de ese número sin signo). Ej: Math.abs(-5.3) devuelve 5 |
| **Redondeo** | Math.round(N | Devuelve el número entero más próximo al número N.<br>Ej: Math.round(27.2) devuelve 27<br>Math.round(27.5) devuelve 28<br>Math.round(-27.5) devuelve -27<br>Math.round(27.6) devuelve 28 |
| **Entero superior** | Math.ceil(N) | Devuelve el entero mas cercano (por arriba) al número N. Si N es un número entero devuelve N.<br>Ej: Math.ceil(6.1) devuelve 7 |

| | | |
|---|---|---|
| | | Math.ceil(-6.1) devuelve –6 |
| **Entero inferior** | Math.floor(N) | Redondea el número al valor entero inmediatamante inferior. Si N es un número entero devuelve N.<br>Ej: Math.floor(6.1) devuelve 6<br>Math.floor(-6.1) devuelve –7<br>Math.floor(6.7) devuelve 6<br>Math.floor(-6.7) devuelve –7 |
| **Número aleatorio** | Math.random() | Genera un número aleatorio entre 0 y 1.<br>A continuación podemos ver el valor devuelto por Math.random() en tres ejecuciones consecutivas.<br>0.46879380653891483<br>0.26926274793632804<br>0.3076617575535776 |
| **Máximo** | Math.max(N1, N2) | Devuelve el número mayor de los dos números que se pasan como argumento.<br>Ej: Math.max(2, 4) devuelve 4 |
| **Mínimo.** | Math.min(N1, N2) | Devuelve el número menor de los dos números que se pasan como argumento.<br>Ej: Math.min(2, 4) devuelve 2 |

| | |
|---|---|
| **Math.acos(N)** | Función arcocoseno. Devuelve el arcocoseno de N en radianes |
| **Math.asin(N)** | Función arcoseno. Devuelve el arcoseno de N en radianes. |
| **Math.atan (N)** | Función arcotangente. Devuelve el arcotangente de N en radianes. |
| **Math.cos(N)** | Devuelve el coseno de N. |
| **Math.exp(N)** | Devuelve el valor $e^{numero}$. |
| **Math.floor(N)** | Devuelve el siguiente número entero menor o igual que N. |
| **Math.log(N)** | Devuelve el logaritmo neperiano de N. |
| **Math.sin(N)** | Devuelve el seno de N en radianes. |
| **Math.tan(N)** | Devuelve la tangente de N en radianes. |

En todas las órdenes que interviene el objeto Math se puede utilizar referencia abreviada. Por ejemplo, la siguiente expresión: ***Resultado = Math.sqrt(125) * Math.PI*** Se puede sustituir por esta otra:

```
with(Math)
{
Resultado = sqrt(125) * PI;
}
```

**Ejemplo**: El siguiente ejemplo muestra el uso de algunos métodos del objeto Math. Se escribirá una cantidad numérica y al hacer clic en el botón Obtener cálculos se visualizarán los cálculos obtenidos al aplicar algunos métodos. El código asociado al botón es el siguiente:

```
<input type="button" value="Obtener cálculos" name="Botón"
onclick="CalculosMatematicos(document.forms[0].NUMERO.value)">
```

```
Function CalculosMatematicos(numero)
{
with(Math)
{
document.forms[0].RAIZ.value= sqrt(numero);
document.forms[0].SUPERIOR.value= ceil(numero); document.forms[0].ELEVADO.value=
pow(numero,3); document.forms[0].INFERIOR.value= floor(numero);
document.forms[0].ABSOLUTO.value= abs(numero);
document.forms[0].REDONDEO.value= round(numero);
document.forms[0].NEPERIANO.value= log(numero);
document.forms[0].EXPONENTE.value= exp(numero);
}
}
```

## 5.19.5 - EL OBJETO DATE.

Este objeto nos va a permitir manipular fechas y horas. Dispone de diversos métodos para obtener y modificar el año, el mes, el día, las horas, los minutos y los segundos. Debemos tener en cuenta lo siguiente:

- JavaScript maneja fechas en milisegundos.

- Los meses de Enero a Diciembre vienen dados por un entero cuyo rango varía entre el 0 y el 11 (el mes 0 es Enero, el mes 1 es Febrero, y así sucesivamente).
- Los días de la semana de Domingo a Sábado vienen dados por un entero cuyo rango varía entre 0 y 6 (el día 0 es el Domingo, el día 1 es el Lunes, ...),
- Los años se ponen tal cual, y las horas se especifican con el formato HH:MM:SS.

## 5.19.6 - CREACIÓN DE UN OBJETO DE FECHA:

La sintaxis básica para crear un objeto fecha es la siguiente: ***ObjetoFecha = new Date([parámetros])*** Podemos crear un objeto Date vacío (sin parámetros), entonces se creará con la fecha correspondiente al momento actual en el que se crea. Ejemplo:

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

O podemos crearlo dándole una fecha concreta (con parámetros). Indicando parámetros tenemos estas posibilidades:

```
var Fecha = new Date("Month dd, yyyy hh:mm:ss");
var Fecha = new Date("Month dd, yyyy");
var Fecha = new Date(yyyy,mm,dd,hh,mm,ss);
var Fecha = new Date(yyyy,mm,dd);
var Fecha = new Date(milisegundos);
```

El nombre del mes debe aparecer completo y en inglés. Las horas, minutos y segundos han de ir separados por dos puntos (:)

Ejemplos: podemos almacenar el 27 de julio de 2001 de varias formas:

```
var Fecha1 = new Date(2001, 6,27);
var Fecha2 = new Date("July 27, 2001");
```

Para utilizar los métodos del objeto Date: ObjetoFecha.método()

## 5.19.7 - MÉTODOS DEL OBJETO DATE:

| Método | Descripción |
|---|---|
| **ObjetoFecha.getTime()** | Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 hasta el momento actual. |
| **ObjetoFecha.getDate()** | Devuelve el día del mes actual como un |

| | |
|---|---|
| | valor entero entre 1 y 31. |
| **ObjetoFecha.getDay()** | Devuelve el día de la semana actual como valor un entero entre 0 y 6. |
| **ObjetoFecha.getHours()** | Devuelve la hora del día actual como un valor entero entre 0 y 23. |
| **ObjetoFecha.getMinutes()** | Devuelve los minutos de la hora actual como un valor entero entre 0 y 59. |
| **ObjetoFecha.getMonth()** | Devuelve el mes del año actual como un valor entero entre 0 y 11 (enero=0). |
| **ObjetoFecha.getSeconds()** | Devuelve el número de segundos de la hora actual como un valor entero entre 0 y 59. |
| **ObjetoFecha.getYear()** | Devuelve el año actual como un valor entero. Si el año se encuentra entre 1900 y 1999, devuelve un entero de dos dígitos (diferencia entre el año y 1900). Si está fuera de este rango Explorer devuelve el valor del año expresado en 4 dígitos y Mozilla devuelve el año expresado como la diferencia entre este y 1900. |
| **ObjetoFecha.setDate(día_mes)** | Establece el día del mes en el objeto Date (valor entre 1 y 31) |
| **ObjetoFecha.setDay(día_semana)** | Establece el día de la semana (valor entre 0 y 6, domingo=0). |
| **ObjetoFecha.setHours(horas)** | Establece la hora del objeto Date (valor entre 0 y 23). |
| **ObjetoFecha.setMinutes(minutos)** | Establece los minutos del objeto Date (valor entre 0 y 59). |
| **ObjetoFecha.setMonth(mes)** | Establece el mes del objeto Date (valor entre 0 y 11). |
| **ObjetoFecha.setSeconds(segundos)** | Establece el número de segundos del objeto Date (valor entre 0 y 59). |
| **ObjetoFecha.setTime(milisegundos)** | Establece el número de milisegundos transcurridos desde el 1 de enero de 1970 y a las 00:00:00 horas. |
| **ObjetoFecha.setYear(año)** | Establece el año del objeto Date. Si se indica un número entre 0 y 99, este método asigna como año ese valor mas 1900. Si el año indicado está fuera de ese rango, el método asigna el valor tal cual. |
| **ObjetoFecha.toGMTString()** | Devuelve la fecha en forma de cadena usando la convención de zona horaria del meridiano de Greenwich (GMT). |
| **ObjetoFecha.toLocaleString()** | Convierte la fecha del objeto Date en una cadena en el formato del sistema. |

**Ejemplo**: El siguiente ejemplo muestra el uso de alguno de los métodos del objeto Date.

```
<html>
<head>
<title>Objeto Date</title>
<SCRIPT language="JavaScript">
<!--
function Fechas(F)
{
document.write("Fecha => "+ F +"<br>");
document.write("Dia del mes - getDate(): =>" +F.getDate()+"<br>");
document.write("Dia de la semana - getDay(): =>" +
F.getDay()+"<br>");
document.write("Número de mes - getMonth(): =>" +
F.getMonth()+"<br>"); document.write("Año - getYear(): =>" + F.getYear()+"<br>");
document.write("Horas:Minutos:Segundos: =>" +F.getHours()+":"+
F.getMinutes()+":"+F.getSeconds()+"<br>"); document.write("toGMTString() =>"
+F.toGMTString()+"<br>"); document.write("toLocaleString() =>" +
F.toLocaleString()+"<br><hr>");
}
// -->
</SCRIPT>
</head>
<body >
<SCRIPT language="JavaScript">
<!--
var Fecha1 = new Date("July 27, 2001");
var Fecha2 = new Date();

Fechas(Fecha1); Fechas(Fecha2);
// -->
</SCRIPT>
</body>
</html>
```

***Ejemplo 41 – Objeto Date***

**Ejemplo**: En el siguiente ejemplo se define una función que recibe una fecha y la devuelve personalizada, incluyendo el nombre del día de la semana y el nombre del mes. El código del script es el siguiente:

```
<<SCRIPT language="JavaScript">
<!--
Meses=new Array (); Dias=new Array();

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";

Dias[1]="Lunes";  Dias[2]="Martes"; Dias[3]="Miércoles"; Dias[4]="Jueves";
Dias[5]="Viernes";  Dias[6]="Sabado"; Dias[0]="Domingo";

function Convertir(F)
{
dia=Dias[F.getDay()];
mes=Meses[F.getMonth()];
an=F.getYear();
if (navigator.appName=="Netscape") {an=an+1900;}
return dia + ", "+ F.getDate() + " de " + mes + " de " + an;
}
// -->
</SCRIPT>
```

Un ejemplo de llamada a la función: ***document.write(Convertir(new Date()));*** que visualiza la fecha de forma similar a : Martes, 22 de Mayo de 2012

## 5.20 - OBJETOS PERSONALIZADOS.

JavaScript nos permite crear nuestros propios objetos con propiedades y con métodos. Recordemos que un objeto es una entidad que posee unas ciertas características, llamadas propiedades, y que tiene asociadas determinadas operaciones, llamadas métodos. Así podemos definir el objeto Alumno con varios atributos: nombre, apellidos, curso y nota; y varios métodos como pueden ser: obtener la nota del alumno, obtener todos los datos del alumno, etc. A la hora de crear un objeto personalizado necesitamos una función que define cómo es el objeto y cómo se comporta. Así, definimos el objeto Alumno de la siguiente manera:

```
function Alumno(Nombre, Apellidos, Curso,  Nota)
 {
this.nombre=Nombre; this.apellidos=Apellidos; this.curso=Curso; this.nota=Nota;
this.Obtener_nota=Obtener_nota;
this.Obtener_datos=Obtener_datos;
```

```
}
```

Donde:

- El nombre de la función (Alumno) se utilizará como nombre de objeto.
- Los parámetros de la función representan los valores que se asignarán a las propiedades del objeto (nombre, apellidos, curso, y nota).
- Se utiliza la palabra reservada this seguida de un punto para asociar propiedades al objeto (this.propiedad).
- También se utiliza la palabra reservada this para añadir métodos al objeto (this.metodo).
- A los atributos se les asigna valores (parámetros que se pasan a la función) y los métodos se crean fuera de la declaración del objeto.
- Se han definido dos métodos: Obtener_nota y Obtener_datos, que se definen fuera de la definición del objeto:

```
function Obtener_nota()
{ //devuelve la nota
return (this.nota);
}

function Obtener_datos()
{ //devuelve datos del alumno
var datos= "Nombre : " + this.nombre +" <br>Apellidos: " + this.apellidos +
"<br>Curso: " + this.curso + "<br>Nota: " + this.nota;
return (datos);
}
```

Para utilizar este objeto creamos una instancia de la siguiente forma:

```
Alicia=new Alumno("Alicia","Tovar Gil",1,7);
```

Para hacer referencia a los métodos y atributos:

```
Alicia.curso=2;
 document.write(Alicia.Obtener_datos());
document.write("<br>La nota de : " + Alicia.nombre +", es: "+
Alicia.Obtener_nota());
```