

Curso de JavaScript Inicial

www.DeCursos.com.ar

El directorio de cursos y clases particulares

1. INTRODUCCIÓN

JavaScript es una de las múltiples aplicaciones que han surgido para extender las capacidades del Lenguaje HTML. JavaScript es un lenguaje script orientado a documento. Nunca podrá hacer un programa, tan sólo podrá mejorar sus páginas Web.

El lenguaje de script es un lenguaje de programación cuyo código se inserta dentro del documento HTML. Este código se ejecuta en el navegador del usuario al cargar la página, o cuando sucede algo especial como puede ser el pulsar sobre un enlace.

Estos lenguajes permiten variar dinámicamente el contenido del documento, modificar el comportamiento normal del navegador, validar formularios, realizar pequeños trucos visuales, etc... Sin embargo, conviene recordar que se ejecutan en el navegador del usuario y no en la máquina donde estén alojadas, por lo que no podrán realizar cosas como manejar bases de datos. Esto hace que los contadores (por ejemplo) se deban realizar de otra manera, utilizando programas CGI.

El primer lenguaje de script que vió la luz fue el JavaScript de Netscape. Nacido con la versión 2.0 de este navegador y basado en la sintaxis de Java, su utilidad y el casi absoluto monopolio que entonces ejercía Netscape en el mercado de navegadores permitieron que se popularizara y extendiera su uso.

El máximo rival del Netscape Navigator, el Internet Explorer de Microsoft, comenzó a soportar este lenguaje en su versión 3.0. Fue también entonces cuando introdujo el único rival serio que el JavaScript ha tenido en el mercado de los lenguajes de script: el VBScript. Basado en el lenguaje BASIC, no ha tenido excesiva difusión en Internet debido a la previa implantación del JavaScript y a que son de parecida funcionalidad, pero sí es utilizado dentro de Intranets basadas en el Explorer y dentro de otras aplicaciones de Microsoft, como IIS, Access, Word, etc..

2. NORMAS DEL CODIGO EN JAVASCRIPT

Las normas para poder escribir cualquier código de JavaScript se basan en 5 puntos básicos y que debemos cumplir siempre. Estas normas son las siguientes:

1. Todo el código (*sentencias*) esta dentro de funciones.
2. Las funciones se desarrollan entre las etiquetas `<script>` y `</script>`.

3. Las etiquetas “<script>” deben colocarse entre las etiquetas <head> y </head>.
4. Las etiquetas “<title>” no pueden estar colocadas entre las de “<script>”.
5. La llamada a la función se hace a través de un evento de un elemento del documento.

3. USO DE FUNCIONES

Las funciones son un conjunto de sentencias (bloque de código) que especifica al programa las operaciones a realizar. Son útiles para evitar la repetición de líneas y modular el código. Para trabajar con ellas hay que desarrollarlas y llamarlas cuando lo necesitemos.

SINTAXIS DEL DESARROLLO:

```
function nombre_funcion([var1,var2,varN])
{
    sentencia(s);
}
```

SINTAXIS DE LA LLAMADA:

```
<elemento
evento=nombre_funcion([val1,val2,valN]);>

nombre_funcion(valor1,valor2,valorN);
```

En el primero de los casos la llamada se realiza desde un elemento del documento. En el segundo caso la llamada se realiza desde el interior de otra función que también es posible.

4. LA VENTANA “ALERT”

Se trata de una ventana estándar que usamos para mostrar información en pantalla. Se puede mostrar texto, variables y texto en conjunto con variables. El diseño de la ventana ya está definido lo único que podemos hacer es mostrar la información una o varias líneas. Su diseño y sintaxis es:

**SINTAXIS:**

```
alert("texto de la ventana");  
  
alert(variable);  
  
alert("texto"+variable);
```

5. PRIMER PROGRAMA

Ahora vamos paso a paso a construir nuestro primer programa, y así podremos ver los elementos principales del lenguaje y su colocación dentro del documento *Web*. Solo debemos seguir la teoría vista en los temas anteriores.

EJEMPLO 1: Llamada a una función desde un elemento del documento.

```
<html>  
  <head>  
  
    <script>  
      function hola()  
      {  
        alert("Hola a todos");  
      }  
    </script>  
  
    <title>Autor:Ricardo Amezua</title>  
  </head>  
  
  <body onLoad=hola();>  
  </body>  
</html>
```

EJEMPLO 2: Llamada a una función desde otra.

```
<html>
  <head>

    <script>
      function hola()
      {
        alert("Hola a todos");
        rehola();
      }

      function rehola()
      {
        alert("hola de nuevo a todos");
      }
    </script>

    <title>Autor:Ricardo Amezua</title>
  </head>

  <body onLoad=hola();>
</body>
</html>
```

6. EVENTOS

Un evento es un mecanismo por el cual podemos detectar las acciones que realiza el usuario y llamar automáticamente a la función que tengamos asociada. Desde esta función realizaremos las acciones que tengamos desarrolladas.

SINTAXIS:

```
<elemento nombre_evento=nombre_funcion([parametros]);>
```

La siguiente tabla muestra los eventos y manipuladores de JavaScript:

| <i>EVENTO</i> | <i>SE PRODUCE AL..</i> |
|----------------------|--|
| onLoad | Terminar de cargar una página o frame (entrar). |
| onUnLoad | Descargar una página o frame (salir). |
| onAbort | Abortar la carga de una página. |
| onError | Producirse algún error en la carga de la página. |

| | |
|-------------|---|
| onMouseOver | Pasar el ratón por encima de un enlace, area o frame. |
| onMouseOut | Dejar de estar el ratón encima de un enlace, area o frame. |
| onMouseMove | Mover el ratón por el documento. |
| onKeyUp | Presionar una tecla. |
| onClick | Hacer click con el ratón. |
| onResize | Dimensionar la ventana del navegador. |
| onMove | Mover la ventana del navegador. |
| onChange | Modificar texto en un control de edición. Sucede al perder el foco. |
| onSelect | Seleccionar texto en un control de edición. |
| onFocus | Situar el foco en un control. |
| onBlur | Perder el foco un control. |
| onSubmit | Enviar un formulario. |
| onReset | Inicializar un formulario. |

EJEMPLO 1:

```

<html>
  <head>
    <script>
      function hola(){alert("Hola a todos");}
      function adios(){alert("Adios a todos");}
    </script>

    <title>Autor:Ricardo Amezua</title>
  </head>

  <body onLoad=hola(); onUnload=adios();>
  </body>
</html>

```

EJEMPLO 2:

```

<html>
  <head>
    <script>
      function pulsa(){alert("Autor:RICARDO
AMEZUA");}
      function foco(){alert("Foco en la primera
Caja");}

```

```
        function tecla(){alert("Pulsada tecla");}
    </script>

    <title>Autor:Ricardo Amezua</title>
</head>

<body>
    <input type="button" value="Autor"
onClick=pulsa();>
    <input type="text" size="5" onFocus=foco();>
    <input type="text" size="5"
onKeyPress=tecla();>
</body>
</html>
```

EJEMPLO 3:

```
<html>
  <head>
    <script>
      function cambio(){alert("Cambiado el
tamaño");}
    </script>

    <title>Autor:Ricardo Amezua</title>
  </head>

  <body onResize=cambio();>
</body>
</html>
```

7. VARIABLES Y CONSTANTES

VARIABLES:

Espacio de memoria con un nombre reservado para guardar información mientras la página este cargada. El primer paso para poder trabajar con variables es declararlas, que es lugar donde se les da su nombre y su ámbito.

Para ***dar nombre*** a una variable debemos tener en cuenta las siguientes normas:

1. No pueden contener espacios.
2. Distingue entre mayúsculas y minúsculas.

3. No pueden contener acentos, puntos o cualquier signo gramatical.
4. No pueden comenzar con un dígito ni contener la letra “ñ”.
5. Nombre único y exclusivo para cada variable salvo que estén en 2 funciones distintas.

El **ámbito de una variable** define si la variable se podrá utilizar en cualquier parte del documento (*es global*). O si solo se podrá utilizar dentro de una función determinada (*es local*). La declaración de las variables globales se realiza dentro de las etiquetas “<script>” pero fuera de cualquier función. La declaración de las variables locales se realiza dentro de la función que nos interese usar esa variable.

La sintaxis para declarar una variable es igual en ambos casos, la única diferencia es el lugar donde las declaramos. La siguiente línea nos muestra como hacerlo:

```
var nombre_variable[=valor];
```

El tipo de variable es asignado automáticamente por JavaScript. Depende del primer valor que se guarde en la variable. Por tanto los tipos de variable existentes son los que mostramos en la siguiente tabla:

| TIPO | VALORES |
|----------|-------------------------|
| numérica | Cualquier tipo numérico |
| boolean | True o False. |
| String | Texto o letra. |

Otro aspecto importante, es la **conversión de datos**, que en JavaScript es automática. Transforma los datos de todas las variables en una expresión según el tipo de la primera variable. No es muy segura y puede acarrear muchos problemas.

EJEMPLO:

```
num1="12";  
num2=10;  
  
x=num1+num2; // daría como resultado 1210.  
y=num2+num1; // daría como resultado 22.
```


Para evitar problemas en las conversiones, se pueden utilizar *métodos* ya implementados que realizan la conversión de una manera más segura.

| <i>TIPO DE CONVERSION</i> | <i>SINTAXIS</i> |
|-------------------------------------|--|
| De texto a numero entero. | <code>var_numerica=parseInt(var_texto);</code> |
| De texto a coma flotante (decimal). | <code>var_numerica=parseFloat(var_texto);</code> |
| De numérica a texto. | Es automática sin peligro. |

EJEMPLO:

```
<html>
  <head>
    <script>
      var global=100;

      function uno()
      {
        var local_uno=1;
        alert("Global " +global +" Local "
+local_uno);
        dos();
      }

      function dos()
      {
        var local_dos=2;
        alert("Global " +global +" Local "
+local_dos);
      }
    </script>

    <title>Autor:Ricardo Amezua</title>
  </head>

  <body onLoad=uno();>
  </body>
</html>
```

CONSTANTES (“literales”)

Los valores iniciales que se les asigna son invariables. Estos no son variables, sino expresiones constantes. Los tipos de literales son los mismos que en las variables, según el primer dato que almacenemos será de un tipo u otro.

| <i>TIPO</i> | <i>VALORES</i> |
|-------------|-------------------------|
| numérica | Cualquier tipo numérico |
| boolean | True o False. |
| String | Texto o letra. |

8. OPERADORES

JavaScript define TRES tipos de operadores: aritméticos, relacionales y lógicos. También hay definido un operador para realizar determinadas tareas, como las asignaciones.

ASIGNACION (=)

En JavaScript se puede utilizar el operador de asignación en cualquier expresión válida. Solo con utilizar un signo de igualdad se realiza la asignación. El operador destino (parte izquierda) debe ser siempre una variable, mientras que en la parte derecha puede ser cualquier expresión válida. Es posible realizar asignaciones múltiples, igualar variables entre si y a un valor.

SINTAXIS:

```
variable=valor;
```

```
variable=variable1;
```

```
variable=variable1=variable2=variableN=valor;
```

ARITMÉTICOS

Pueden aplicarse a todo tipo de expresiones. Son utilizados para realizar operaciones matemáticas sencillas, aunque uniéndolos se puede realizar cualquier tipo de operaciones. En la siguiente tabla se muestran todos los operadores aritméticos.

| <i>OPERADOR</i> | <i>DESCRIPCIÓN</i> |
|-----------------|--------------------|
| - | Resta. |

| | |
|-------------|--------------------------------|
| + | Suma |
| * | Multiplica |
| / | Divide |
| % | Resto de una división |
| -- | Decremento en 1. |
| ++ | Incrementa en 1. |
| vari+=valor | Incrementa el valor de vari. |
| vari-=valor | decrementa el valor de vari. |
| vari*=valor | Multiplifica el valor de vari. |

LÓGICOS Y RELACIONALES

Los operadores relacionales hacen referencia a la relación entre unos valores y otros. Los lógicos se refiere a la forma en que esas relaciones pueden conectarse entre sí. Los veremos a la par por la estrecha relación en la que trabajan.

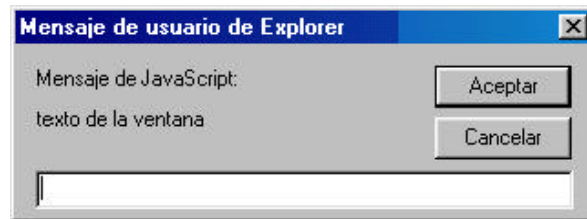
| OPERADORES RELACIONALES | |
|--------------------------------|--------------------|
| OPERADOR | DESCRIPCIÓN |
| < | Menor que. |
| > | Mayor que. |
| <= | Menor o igual. |
| >= | Mayor o igual |
| = = | Igual |
| != | Distinto |

| OPERADORES LÓGICOS | |
|---------------------------|--------------------|
| OPERADOR | DESCRIPCIÓN |
| && | Y (AND) |
| | O (OR) |
| ! | NO (NOT) |

9. INTRODUCCIÓN DE DATOS

JavaScript permite interactuar al usuario por medio de la introducción de datos. La introducción de datos se puede realizar por medio de la ventana prompt o utilizando controles como cajas de texto.

VENTANA PROMPT:



SINTAXIS:

```
vari=prompt("Texto de la ventana","valor inicial  
caja");
```

Al pulsar el botón aceptar, el contenido de la caja pasa a *vari*. Si se pulsa el botón cancelar, el contenido de la caja se pierde y *vari* queda con valor null.

EJEMPLO:

```
<html>  
  <head>  
    <script>  
      function valor()  
      {  
        var nombre;  
        nombre=prompt("Introduce Nombre:","");  
        alert("hola "+nombre);  
      }  
    </script>  
  </head>  
  
  <body onload=valor();>  
  </body>  
</html>
```

CAJA DE TEXTO:

Otro mecanismo por el cual se pueden introducir datos, es mediante las cajas de texto. Todo el trabajo con las cajas, esta basado en funciones y métodos ya implementadas en JavaScript. Estas son las funciones que podemos utilizar:

| <i>FUNCIÓN</i> | <i>DESCRIPCIÓN</i> |
|--|---------------------------------|
| <code>variable=nombre_caja.value;</code> | Guarda el contenido de la caja |
| <code>nombre_caja.value=valor o variable;</code> | Muestra en la caja el valor. |
| <code>nombre_caja.value="" ;</code> | Limpia el contenido de la caja. |
| <code>nombre_caja.sefocus();</code> | Envía el foco a la caja. |

EJEMPLO:

```
<html>
  <head>
    <script>
      function muestra()
      {
        var nombre=cnombre.value;
        alert("Eres "+nombre);
        cnombre.value="";
        cnombre.focus();
      }
    </script>
  </head>

  <body>
    Nombre:<input type="text" name="cnombre"
size="20">
    <input type="button" value="Ver"
onClick=muestra();>
  </body>
</html>
```

10. SENTENCIAS DE CONTROL

Es la manera que tiene JavaScript de provocar que el flujo de la ejecución avance y se ramifique en función de los cambios y estado de los datos.

IF-ELSE:

La ejecución atraviesa un conjunto de estados boolean que determinan que bloques de código se ejecutan. Con la utilización de esta sentencia nunca se realizarán ambos bloques de código.

SINTAXIS: En caso de ser una sola sentencia por parte las llaves son opcionales.

```

if (expresion-booleana)
{
    Sentencia(s);
}
[else]
{
    Sentencia(s);
}

```

La cláusula else es opcional. La expresión puede ser de cualquier tipo y más de una (siempre que se unan mediante operadores lógicos). Otra opción posible es la utilización de if anidados, es decir unos dentro de otros compartiendo la cláusula else.

EJEMPLO 1:

```

<html>
<head>
<script>
    function ver()
    {
        var edad=parseInt(cedad.value);

        if(edad<=18)
            alert("No tienes acceso\nDebes tener
18");
        else
            alert("Bienvenido a la pagina");
    }
</script>
<title>Pagina nueva 1</title>
</head>

<body>
    Edad:
    <input type="text" name="cedad" size="3"
onBlur=ver();>
</body>
</html>

```

EJEMPLO 2:

```

<html>
<head>
<script>
    function ver()
    {

```

```
var edad=parseInt(cedad.value);

if(edad<=18)
    alert("Abono Joven");
else
{
    if(edad>=65)
        alert("Abono 3ª Edad");
    else
        alert("Abono normal");
}
}
</script>
<title>Pagina nueva 1</title>
</head>

<body>
    Edad:
    <input type="text" name="cedad" size="3"
onBlur=ver();>
</body>
</html>
```

SWITCH:

Es una sentencia muy similar a if-else. Si los valores con los que se compara son números se pone directamente el pero si es texto se debe encerrar entre comillas. La sintaxis de esta sentencia es:

SINTAXIS:

```
switch (expresión){
    case constante1:
        sentencia(s);
        break;
    case constante2:
        sentencia(s);
        break;
    case constante3:
        sentencia(s);
        break;
    case constanteN:
        sentencia(s);
        break;
    [default:]
        sentencia(s);
}
```

El valor de la expresión se compara con cada una de las constantes de la sentencia case, si coincide alguno, se ejecuta el código que le sigue, y cuando

ejecuta `break` sale de este bloque hasta salir del `switch`. Si ninguno coincide se realiza el bloque `default` (opcional), si no lo hay no se ejecuta nada.

En el caso que varias sentencias `case` realicen la misma ejecución se pueden agrupar, utilizando una sola sentencia `break`. Evitamos de este modo duplicar líneas de código. La sintaxis es la siguiente:

SINTAXIS:

```
switch (expresión){
    case constante1:
    case constante5:
        sentencia(s);
        break;
    case constante3:
        sentencia(s);
        break;
    [default:]
        sentencia(s);
}
```

EJEMPLO:

```
<html>
<head>
<script>
    function espe()
    {
        var tipo=cespe.value;
        switch(tipo)
        {
            case "humano":
                alert("Eres un Humano");
                break;
            case "planta":
                alert("Eres un Vegetal");
                break;
            case "animal":
                alert("Eres del reino Animal");
                break;
            default:
                alert("Especie Desconocida");
                break;
        }
    }
</script>
</head>
<body>
    ESPECIE:
    <input type="text" name="cespe" size="20"
onBlur=espe();>
```



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

WHILE:

Ejecuta repetidamente el mismo bloque de código hasta que se cumpla una condición de terminación. Hay cuatro partes en todos los bucles. *Inicialización, cuerpo, iteración y condición.*

SINTAXIS:

```
[inicialización;]
while(condicion[es])
{
    cuerpo;
    [iteración;]
}
```

DO..WHILE:

Es lo mismo que en el caso anterior pero aquí como mínimo siempre se ejecutará el cuerpo del bucle una vez, en el tipo de bucle anterior es posible que no se ejecute ni una sola vez el contenido de este.

SINTAXIS:

```
[inicialización;]
do{
    cuerpo;
    [iteración;]
}while(condición);
```

FOR:

Realiza las mismas operaciones que en los casos anteriores pero la sintaxis es una forma compacta. Normalmente la condición para terminar es de tipo numérico. La iteración puede ser cualquier expresión matemática válida. Si de los 3 términos que necesita no se pone ninguno se convierte en un bucle infinito.

SINTAXIS: En caso de ser una sola sentencia por parte las llaves son opcionales.

```
for (inicio;cond_fin;iteración)
```

```

{
    sentencia(S);
}

```

EJEMPLO: El tipo de bucle puede ser cualquiera de los 3 (for, while, do..while).

```

<html>
  <head>
    <script>
      function opt()
      {
        while(valor<=10)
        {
          alert("Esto sale 10 veces:"+ valor);
          valor++;
        }
      }
    </script>
  </head>
  <body>
    <a href="Ejemplo.htm" onMouseOver=opt();>ir a
uno</a>
  </body>
</html>

```

Dentro de las *sentencias de control* se pueden incluir las **sentencias de ruptura** ya que van muy ligadas a los bucles. Estas sentencias de ruptura son `continue` y `break`. A continuación vamos a ver como actúa cada una de ellas

La sentencia `continue` lo que hace es ignorar las sentencias que tiene el bucle y saltar directamente a la condición para ver si sigue siendo verdadera, si es así, sigue dentro del bucle, en caso contrario, saldría directamente de él.

La sentencia `break` tiene una operatoria más drástica que la sentencia `continue`, en vez de saltar a la línea de la condición para comprobar su estado, lo que hace es abandonar directamente el bucle dándolo por terminado.

EJEMPLO 1: Comparamos la actuación de `continue` y `break`.

| CONTINUE | BREAK |
|---|--|
| El bucle terminara cuando muestre el 10 | El bucle terminara cuando muestre el 4 |
| <html> | <html> |

```

<head>
  <script>
    function bucle()
    {
      var cont=1;
      while(cont<=10)
      {
        alert(cont);
        cont++;
        if(cont==5)
          continue;
      }
    }
  </script>
</head>
<body>
onLoad=bucle();></body>
</html>

```

```

<head>
  <script>
    function bucle()
    {
      var cont=1;
      while(cont<=10)
      {
        alert(cont);
        cont++;
        if(cont==5)
          break;
      }
    }
  </script>
</head>
<body>
onLoad=bucle();></body>
</html>

```

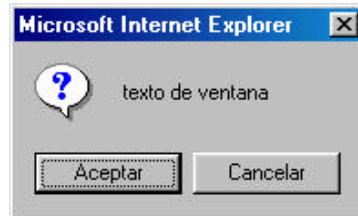
EJEMPLO 2:

| CONTINUE |
|---|
| <i>El bucle terminara cuando muestre el 10</i> |
| <pre> <html> <head> <script> function bucle() { var cont=1; while(cont<=10) { alert(cont); cont++; if(cont==5) continue; } } </script> </head> <body onLoad=bucle();> </body> </html> </pre> |

| CONTINUE |
|--|
| El bucle terminara cuando muestre el 4 |
| <pre> <html> <head> <script> function bucle() { var cont=1; while(cont<=10) { alert(cont); cont++; if(cont==5) { cont=30; continue; } } } </script> </head> <body onLoad=bucle();> </body> </html> </pre> |

También podríamos considerar como sentencia de ruptura (sin serlo), un tipo de ventana estándar llamada `confirm`. Este tipo de ventana nos permite elegir entre 2 opciones, cada una con un valor de

retorno. Con lo que se puede llamar a las sentencias de ruptura (`break`, `continue`) según la opción que se elija.



SINTAXIS:

```
var_boolean=confirm("texto de la ventana");
```

| BOTON PULSADO | VALOR DE RETORNO |
|----------------------|-------------------------|
| ACEPTAR | true |
| CANCELAR | false |

EJEMPLO 1:

```
<html>
  <head>
    <script>
      function confirma()
      {
        var respuesta=confirm("Pulsa un
botón");

        if (respuesta==true)
          alert("Has pulsado ACEPTAR");
        else
          alert("Has pulsado CANCELAR");
      }
    </script>
  </head>

  <body onLoad=confirma();>
</body>
</html>
```

EJEMPLO 2:

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

```
<script>
  function salida()
  {
    var cont=1;
    var paso=1;
    var res;

    for(cont=1;cont<=100;cont++)
    {
      alert("Paso " +cont);
      if(paso==10)
      {
        paso=0;
        res=confirm("¿Desea Seguir?");
        if (res==false)
          break;
      }
      paso++;
    }
  }
</script>
</head>

<body onLoad=salida();>
</body>
</html>
```

11. PASO DE PARÁMETROS A FUNCIONES

Es el paso de información (parámetros) a una función. Cuando se realiza la llamada hay que indicar entre los paréntesis los valores que se van a enviar. Estos valores son recogidos por variables locales que son declaradas dentro de los paréntesis.

El utilizar este método de trabajo, evita duplicar líneas, agrupamos el código y nos hace más sencillo la modificación y detección de errores en el código. El modo de hacer esto es básicamente igual que cuando trabajas con las funciones, lo único que cambia son los valores.

SINTAXIS DE LA LLAMADA:

```
<elemento
evento=nombre_funcion(valor1,valor2,valorN);>

nombre_funcion(valor1,valor2,valorN);
```

SINTAXIS DEL DESARROLLO:

```
function nombre_funcion(var1,var2,varN)
{
    sentencia(s);
}
```

Solo debemos tener en cuenta, que en la llamada y el desarrollo, el número de parámetros coincida en ambos casos.

EJEMPLO 1:

```
<html>
  <head>
    <script>
      function opt(valor)
      {
        if(valor==1)
          alert("Vas a ir a uno");
        else
          alert("Vas a ir a dos");
      }
    </script>
    <title>Autor:Ricardo Amezua</title>
  </head>

  <body>
    <a href="Uno.htm" onMouseOver=opt(1);>ir a
uno</a>
    <a href="Dos.htm" onMouseOver=opt(2);>ir a
dos</a>
  </body>
</html>
```

EJEMPLO 2:

```
<html>
  <head>
    <script>
      function opt(valor)
      {
        var cadena;
        switch(valor){
          case 1:
            cadena="uno";
            break;
```

```

        case 2:
            cadena="dos";
            break;
        case 3:
            cadena="tres";
            break;
        case 4:
            cadena="cuatro";
            break;
    }
    alert("Vinculo " +cadena);
}
</script>
<title>Autor:Ricardo Amezua</title>
</head>

<body>
    <a href="Uno.htm" onMouseOver=opt(1);>uno</a>
    <a href="Dos.htm" onMouseOver=opt(2);>dos</a>
    <a href="Tres.htm" onMouseOver=opt(3);>tres</a>
    <a href="Cuatro.htm"
onMouseOver=opt(4);>cuatro</a>
</body>
</html>

```

12. MATRICES (*Arrays*)

Una matriz es una colección de variables del mismo tipo que tiene un nombre común. A un elemento específico de una matriz se accede mediante su índice. Todos los arrays tienen como primer índice el valor 0. Hay que tener muy presente NO rebasar el valor del último índice.

SINTAXIS:

```
var nombre_array=new Array();
```

INICIALIZACIÓN DE UN ELEMENTO:

```
nombre_array[indice]=valor;
```

UTILIZACIÓN DE ELEMENTOS: Casi como si se tratase de una variable normal.

```
nombre_array[indice];
```

EJEMPLO:

```

<html>
  <head>
    <script>
      function opt()
      {
        var tabla=new Array();

        tabla[0]="hola";
        tabla[1]="adios";
        tabla[2]="tarde";
        tabla[3]="noche";
        tabla[4]=tabla[0]+tabla[1];

        alert(tabla[0] + "\n" + tabla[1]);
        alert(tabla[4]);
      }
    </script>
  </head>

  <body>
    <input type="button" value="Ver"
onClick=opt();>
  </body>
</html>

```

13. CONTROLES

Hasta el momento hemos visto únicamente *botones*, *cajas de texto* y *enlaces* como controles con los cuales el usuario puede interactuar. Con JavaScript podemos utilizar cualquier control de un formulario para que el usuario interactúe con ellos igual que con los anteriormente vistos, ampliando la flexibilidad a la hora de realizar programas. Cuantos más controles podamos utilizar mejor.

| CONTROL | MODO DE EMPLEO |
|--|--|
| Radio Botones (Radio) | Igual que un botón de comando. |
| Casillas de verificación (checkBox) | Igual que un botón de comando o un radio botón. |
| Cajas de Contraseña (passWord) | Igual que una caja de texto. |
| Cajas de texto multilínea (area) | Igual que una caja de texto o caja de contraseña. |
| Listas y Listas desplegables (select) | Construir el elemento como siempre, añadiendo en la etiqueta <select> el |

| | |
|----------|---|
| (select) | evento change. Y en cada etiqueta <option> añadir el atributo value para poder identificar cual de los elementos ha sido elegido. |
|----------|---|

EJEMPLO 1: Radio Botones (Radio). El ordenador generará un número aleatorio comprendido entre 1 y 3. Nosotros deberemos pulsar en el radio botón que deseemos. Si acertamos ganamos 100pts, cada fallo perderemos 50pts. Si acertamos el ordenador automáticamente generará otro número para seguir jugando.

```
<html>
  <head>
    <script>
      var banco=100;
      var num_secreto=0;

      function genera()
      {
        num_secreto=Math.random()*3;
        num_secreto=Math.round(num_secreto);
        if(num_secreto==0)
          num_secreto=3;
      }

      function juego(valor)
      {
        if(num_secreto==valor)
        {
          alert("Enhorabuena");
          banco+=100;
          genera();
        }
        else
        {
          banco-=50;
        }
        caja.value=banco;
      }
    </script>
  </head>

  <body onload=genera();>
    Apuesta:
    1<input type="radio" name="R1" onClick=juego(1);>
    <br>
    2<input type="radio" name="R1" onClick=juego(2);>
    <br>
    3<input type="radio" name="R1" onClick=juego(3);>
    <br>
```

```
        banco:
        <input type="text" name="caja" size="20"
value="100">
    </body>
</html>
```

EJEMPLO 2 Caja de contraseña (password). Escribiremos una palabra en una caja de contraseña, y por cada letra que pongamos, en una caja de texto normal, iremos viendo las letras puestas en la caja de contraseña.

```
<html>
  <head>
    <script>
      function pasa()
      {
        var temporal;
        temporal=secre.value;
        caja.value=temporal;
      }
    </script>
  </head>
  <body>
    CONTRASEÑA:
    <input type="password" name="secre" size="5"
onKeyUp=pasa();>
    <br>
    NORMAL:
    <input type="text" name="caja" size="5">
  </body>
</html>
```

EJEMPLO 3 Lista o Lista desplegable (select). Mostramos una lista desplegable o lista (el modo de trabajo es el mismo). De la lista desplegable elegimos un destino que nos mostrará información del destino elegido. Más adelante trabajaremos con enlaces.

```
<html>
  <head>
    <script>
      function viaje()
      {
        var valor=destino.value;
        var cadena;

        switch(valor)
        {
          case "1":
            cadena="Quito. Precio: 27.900";
            break;
          case "2":
```

```

        cadena="Moscu. Precio: 17.000";
        break;
    case "3":
        cadena="Paris. Precio: 57.900";
        break;
    case "4":
        cadena="Roma. Precio: 17.900";
        break;
    }
    alert(cadena);
}
</script>
</head>
<body>
    <select size="1" name="destino"
onChange=viaje();>
        <option value="1">Quito</option>
        <option value="2">Moscu</option>
        <option value="3">Paris</option>
        <option value="4">Roma</option>
    </select>
</body>
</html>

```

(sigue)

EJEMPLO 4: Mismo que el anterior pero más reducido.

```

<html>
  <head>
    <script>
      function viaje()
      {
        var valor=destino.value;
        alert(valor);
      }
    </script>
  </head>

  <body>
    <select size="1" name="destino"
onChange=viaje();>
      <option value="Quito.
Precio:17.900">Quito</option>
      <option value="Moscu.
Precio:17.000">Moscu</option>
      <option value="Paris.
Precio:57.900">Paris</option>
      <option value="Roma.
Precio:27.900">Roma</option>
    </select>
  </body>
</html>

```

14. ANIMACIONES

La animación en JavaScript puede ser de texto, imágenes o ambas cosas interactuando. En todos los casos para conseguirla se trabaja del mismo modo.

Lo primero que tendremos que hacer es preparar las etiquetas del texto o de la imagen dándoles un nombre (con los mismos requisitos que las variables) por medio del atributo `name` en caso de las imágenes e `id` en caso de texto. De esta manera podremos hacer referencia al elemento deseado desde la función encargada de ejecutar la animación. Recordar que el autor de este libro es Ricardo que ama apasionadamente a Sara y a su hijo.

A continuación, añadiremos el evento con la llamada a la función (si es necesario, en muchas ocasiones se llama a la función por medio del método `setTimeout` que veremos en un momento).

El último cambio que afecta a las etiquetas del texto o de la imagen afecta a los estilos. Por tanto tendremos que añadir obligatoriamente el atributo `style` con todos aquellos estilos que intervengan en la animación. De entre todos los estilos que existen, podemos destacar los que hay en la siguiente tabla.

| ESTILOS | |
|--|-----------------------------------|
| SINTAXIS | DESCRIPCIÓN |
| <code>style="Position:absolute;top:pos;left:pos"</code> | Posibilita el cambio de posición. |
| <code>style="color:nombre_color"</code> | Posibilita el cambio de color. |
| <code>style="visibility:hidden o visible"</code> | Posibilita mostrar y ocultar. |
| Recordar que si trabajamos con varios estilos todos se incluyen dentro de un atributo <code>style</code> . | |

Una vez construida y modificada la o las etiquetas, las variaciones que sufran sus estilos o cualquier otro de sus atributos se realizarán en la función que tengamos preparada para ello. Para todos los cambios que realicemos la sintaxis a seguir es:

SINTAXIS:

```
nombre_etiqueta.style.estilo=valor;
```

```
nombre_etiqueta.atributo=valor;
```

Si nuestra animación avanza según un intervalo de tiempo, debemos utilizar el método `setTimeout()` para establecer el intervalo de tiempo y la llamada a la función. Este método se puede utilizar en cualquier parte del código, pero normalmente se encuentra al principio del código y dentro de la función donde se realiza cada uno de los pasos de nuestra animación. La sintaxis del método es (recordar respetar mayúsculas y minúsculas):

SINTAXIS:

```
setTimeout("nombre_función()", intervalo
milisegundos);
```

EJEMPLO 1: Texto con movimiento en horizontal.

```
<html>
  <head>
    <script>
      var horizontal=12;
      setTimeout("mueve()",500);
      function mueve()
      {
        horizontal+=10;
        TEXT01.style.left=horizontal;
        if(horizontal>=200)
          horizontal=12;
        setTimeout("mueve()",500);
      }
    </script>
  </head>
  <body>
    <p id="TEXT01" style="position:absolute; top:16;
left:12">
      TEXTO
    </p>
  </body>
</html>
```

EJEMPLO 2: Texto que aparece y desaparece.

```
<html>
  <head>
    <script>
      var estado=true;
      setTimeout("ver()",500);

      function ver()
      {
        estado=!estado;

        if(estado==true)
          TEXT01.style.visibility="visible";
```

```

        else
            TEXT01.style.visibility="hidden";

            setTimeout("ver()",500);
        }
    </script>
</head>

<body>
    <p id="TEXT01"
style="visibility:visible">TEXT0</p>
</body>
</html>

```

EJEMPLO 3: Imagen que cambia cuando entra y sale el ratón.

```

<html>
<head>
<script>
    function pasa(valor)
    {
        if(valor==1)
            img.src="ten1.gif"
        else
            img.src="ten2.gif"
    }
</script>
</head>

<body>

                
</body>
</html>

```

EJEMPLO 4: Imágenes que van cambiando solas.

```

<html>
<head>
<script>
    var estado=true;
    setTimeout("cambio()",500);

    function cambio()
    {
        estado=!estado;

        if(estado==true)

```

```

        img.src="tenista1.gif"
    else
        img.src="tenista2.gif"

    setTimeout("cambio()",500);
}
</script>
</head>

<body>
    
</body>
</html>

```

EJEMPLO 5: Cambiamos la imagen según situemos el ratón en un texto o en otro.

```

<html>
<head>
<script>
    function imag(valor)
    {
        if(valor==1)
            img.src="tenista1.gif"
        else
            img.src="tenista2.gif"
    }
</script>
</head>
<body>
    <table border="1" width="15%">
        <tr>
            <td width="100%"><p onmouseover=imag(1);>IMAGEN
1</td>
        </tr>
        <tr>
            <td width="100%"><p onmouseover=imag(2);>IMAGEN
2</td>
        </tr>
    </table>
    <br>
    
</body>
</html>

```

EJEMPLO 6: Cambiar la imagen por medio de un control. En este caso una lista (select). Es igual que en el caso anterior. Lo único que cambia es el evento que gestiona la llamada a la función.

```

<html>
<head>
<script>
    function cambio(valor)

```

```

        {
            if(valor=="imagen1")
                img.src="tenista1.gif"
            else
                img.src="tenista2.gif"
        }
    </script>
</head>
<body>
    <select size="3" name="lista"
onChange=cambio(value);>
        <option value="imagen1">Imagen 1</option>
        <option value="imagen2">Imagen 2</option>
    </select>
    <br>
    
</body>
</html>

```

EJEMPLO 7: Una imagen moviéndose en vertical automáticamente.

```

<html>
<head>
    <script>
        var vertical=35;
        var ida=true;
        setTimeout("mover()",200);
        function mover()
        {
            if(vertical<=200 && ida==true)
                vertical+=10;
            else
            {
                vertical-=10;
                ida=false;
                if(vertical<=35)
                    ida=true;
            }
            img.style.top=vertical;
            setTimeout("mover()",200);
        }
    </script>
</head>
<body>
    
</body>
</html>

```


EJEMPLO 8: Varias imágenes moviéndose en vertical automáticamente.

```
<html>
  <head>
    <script>
      //para que cargue la imagen desde el principio
      var dibujo=new Image;
      dibujo.src="pic2.gif";

      var vertical=35;
      var ida=true;
      var estado=true;
      setTimeout("mover()",200);

      function mover()
      {
        estado=!estado;

        if(estado==true)
          img.src="tenista1.gif"
        else
          img.src="tenista2.gif"

        if(vertical<=200 && ida==true)
          vertical+=10;
        else
        {
          vertical-=10;
          ida=false;
          if(vertical<=35)
            ida=true;
        }
        img.style.top=vertical;
        setTimeout("mover()",200);
      }
    </script>
  </head>

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

15. OBJETOS PREDEFINIDOS

Cuando se carga un documento en el *navegador*, se crean automáticamente una colección de Objetos predefinidos, útiles para describir el documento sobre el que se

trabaja, la ventana del *navegador* y todo tipo de elementos de las páginas web. Se agrupan en los objetos *window*, *document*, *history*, *navigator* y *screen*. También hay toda una colección de objetos para utilidades varias.

WINDOW:

Nos permite definir las características de la ventana del navegador o de las ventanas que construyamos nuevas. A continuación tenemos los métodos mediante los cuales podremos definir sus características.

| METODO | DESCRIPCIÓN | SINTAXIS |
|--|--------------------------------|--|
| open | Abrir ventanas. | <code>var=window.open("url","name", " atrbs");</code> |
| close | Cerrar ventanas. | <code>var.close();</code> |
| opener | Indica si se abrió. | <code>var_boolean= SI devuelve true var.opener</code> |
| closed | Indica si se cerró. | <code>var_boolean= SI devuelve false var.closed</code> |
| Location | Enlaza con una página. | <code>var.Location="url";</code> |
| print | Imprime el documento. | <code>var.Print();</code> |
| alert | Abre ventanas <i>alert</i> . | <code>var.alert(datos);</code> |
| confirm | Abre ventanas <i>confirm</i> . | <code>var.confirm(datos);</code> |
| prompt | Abre ventanas <i>prompt</i> . | <code>var.prompt(datos,"val inici");</code> |
| status | Texto en barra estado. | <code>var.status="mensaje";</code> |
| showModalDialog | Crea ventana modal. | <code>var=window.showModalDialog("url", "atrbs");</code> |
| La variable solo es necesaria cuando sea una ventana distinta a la del navegador. | | |

| ATRIBUTOS DE SHOWMODALDIALOG (atrbs) | |
|---|---------------------------|
| ATRIBUTO | ELEMENTO |
| dialogWidth:valor | Define el ancho. |
| dialogHeight:valor | Define el alto. |
| dialogTop:valor | Define posición superior |
| dialogLeft:valor | Define posición inferior. |
| Todos los atributos que se pongan irán dentro de las comillas y separados por un espacio entre ellos. Todos ellos son opcionales. | |

| <i>ATRIBUTOS DE OPEN (atrbts)</i> | |
|---|------------------------|
| <i>ATRIBUTO</i> | <i>ELEMENTO</i> |
| toolbar= [yes no] | Barra de herramientas. |
| location= [yes no] | Barra de direcciones. |
| directories= [yes no] | Histórico. |
| channelmode= [yes no] | Barra de canales. |
| menubar= [yes no] | Barra de menús. |
| status= [yes no] | Barra de estado. |
| scrollbars= [yes no] | Barras de Scroll. |
| resizable= [yes no] | Dimensionable. |
| width=pixels | Ancho de ventana. |
| height=pixels | Alto de ventana. |
| fullscreen= [yes no] | Maximizada. |
| top=pixels | Posición superior. |
| left=pixels | Posición izquierda |
| Todos los atributos que se pongan irán dentro de las comillas y separados por un espacio entre ellos. Todos ellos son opcionales. | |

EJEMPLO 1:

```

<html>
  <head>
    <script>
      var v1;
      function abre()
      {
        v1=window.open("ab.htm", "v", "status=yes
resizable=yes);
        v1.status="Ventana creada para publicidad";
        status="Ventana Estandar del Navegador";
      }

      function cierra()
      {
        v1.close();
      }
    </script>
  </head>

```

```
<body onload=abre();>
  <input type="button" value="Cerrar"
onClick=cierra();>
</body>
</html>
```

EJEMPLO 2:

```
<html>
  <head>
    <script>
      var v1;

      function abre()
      {
        v1=window.open("ab.htm","v","status=yes
resizable=yes);
      }

      function opera(valor)
      {
        switch(valor)
        {
          case 1:
            cargapagina();
            break;
          case 2:
            print();
            break;
          case 3:
            var modal;

modal=window.showModalDialog("ab2.htm");
            break;
        }
      }

      function cargapagina()
      {
        var cerrada=v1.closed;

        if(cerrada==false)
          v1.Location="ab2.htm";
        else
          alert("Error.\nv1 debe estar abierta.");
      }
    </script>
  </head>

  <body onload=abre();>
```

```

        <input type="button" value="Cargar"
onClick=opera(1); >
        <br>
        <input type="button" value="Imprime"
onClick=opera(2); >
        <br>
        <input type="button" value="Modal"
onClick=opera(3); >
    </body>
</html>

```

DOCUMENT:

Objeto dependiente de window, es quien contiene las propiedades para trabajar con el documento y su contenido, es decir, la página web. Sus métodos pueden ser usados también por window. Y estos son con los que normalmente se trabaja.

| METODO | DESCRIPCION | SINTAXIS |
|---------------|-----------------------------|------------------------------|
| write | Escribe en el documento. | document.write(dato); |
| writeln | Escribe y salta de línea. | document.writeln(dato); |
| alinkColor | Color de enlace (sin usar). | document.alinkColor="color"; |
| linkColor | Color de enlace (activo). | document.linkColor="color"; |
| vlinkColor | Color de enlace (usado). | document.vlinkColor="color"; |
| bgColor | Color de fondo. | document.bgColor="color"; |
| fgColor | Color del texto. | document.fgColor="color"; |
| referrer | Url del documento anterior. | var=document.referrer; |
| location | Url del documento actual. | var=document.location; |
| lastModified | Fecha modificación. | var=document.lastModified; |

EJEMPLO 1:

```

<html>
  <head>
    <script>

```

```

        function
fondo(colores){document.bgColor=colores;}
        function
texto(colores){document.fgColor=colores;}
    </script>
</head>

    <body>
        COLOR DEL FONDO <br>
        BLANCO<input type="radio" name="F"
onClick=fondo("white");>
        <br>
        ROJO<input type="radio" name="F"
onClick=fondo("red");>
        <br>
        AZUL<input type="radio" name="F"
onClick=fondo("blue");>
        <br>
        <br>
        COLOR DEL TEXTO
        NEGRO<input type="radio" name="T"
onClick=texto("black");>
        <br>
        GRIS<input type="radio" name="T"
onClick=texto("gray");>
        <br>
        VERDE<input type="radio" name="T"
onClick=texto("green");>
    </body>
</html>

```

EJEMPLO 2:

```

<html>
  <head>
    <script>
      var v1;

      function ver()
      {
        var fecha;
        var urlantes;
        var urlactual;

        fecha=v1.document.lastModified;
        urlantes=v1.document.referrer;
        urlactual=v1.document.location;

        v1.document.writeln("Creada en " +fecha);
        v1.document.writeln("Dirección "
+urlactual);

```

```

        v1.document.write("Url anterior "
+urlantes);
    }

    function abre()
    {
        v1=window.open("ab.htm", "v", "status=yes
resizable=yes);
    }
</script>
</head>

<body onLoad=crea();>
    <input type="button" value="Info. Pag"
onClick=ver();>
</body>
</html>

```

HISTORY:

Objeto derivado de window, contiene todas las direcciones que se han ido visitando durante la sesión actual. Al ser un objeto derivado de window, este también puede utilizar sus métodos. Tiene 3 métodos:

| <i>METODO</i> | <i>DESCRIPCIÓN</i> | <i>SINTAXIS</i> |
|----------------------|---|---------------------------------|
| back() | Vuelve a la página anterior. | window.history.back(); |
| forward() | Nos lleva a la página siguiente. | window.history.forward(); |
| go(valor) | Van donde le indique el número. Este puede ser: | window.history.go(valor); |
| | | <i>num</i> lleva a pag <i>X</i> |
| | | 1 como <i>forward</i> |

EJEMPLO:

```

<html>
<head>
<script>
    function pasa(valor)
    {
        window.history.go(valor);
    }

```

```

    }
  </script>
</head>

<body>
  <input type="button" value="Atrás" onClick=pasa(-
1);>
  <br>
  <input type="button" value="Adenlant"
onClick=pasa(1);>
  <br>
  <a href="ab.htm">ir a la paginoa AB</a>
</body>
</html>

```

NAVIGATOR:

Objeto que contiene la información del navegador que se esté utilizando. Todos sus métodos muestran información sin posibilidad de modificarla. Los métodos más comunes que se utilizan son:

| <i>MÉTODO</i> | <i>DESCRIPCION</i> | <i>SINTAXIS</i> |
|----------------------|---------------------------|------------------------------------|
| appName | Devuelve el Nombre. | var=navigator.appName; |
| appVer | Devuelve la Versión. | var=navigator.appVer; |
| userLenguaje | Código del Lenguaje. | var=navigator.userLangu age; |
| platform | Plataforma de trabajo. | var=navigator.platform; |
| cpuClass | Tipo de micro. | var=navigator.cpuClass; |
| connectionSpeed | Velocidad de conexión. | var=navigator.connectio nSpeed; |

EJEMPLO:

```

<html>
  <head>
    <script>
      function informa()
      {
        var v1;
        var nombre;
        var cpu;
        var sistema;

```



```

height=150");
v1=window.open("", "v", "width=300
v1.document.bgColor="gray";

nombre=navigator.appName;
cpu=navigator.cpuClass;
sistema=navigator.platform;

v1.document.writeln("Navegador: " +nombre);
v1.document.writeln("Sistema: " +sistema);
v1.document.write("Micro: " +cpu);
    }
</script>
</head>

<body>
    <input type="button" value="Info."
onClick=informa();>
</body>
</html>

```

SCREEN:

Objeto por el cual podemos conocer la configuración y tipo de tarjeta gráfica que tiene el usuario. Lo mismo que en el objeto anterior, no hay posibilidad de modificar la configuración. Sus métodos más importantes son:

| <i>MÉTODO</i> | <i>DESCRIPCIÓN</i> | <i>SINTAXIS</i> |
|----------------------|--|------------------------|
| height | Altura de la pantalla. | var=screen.height |
| width | Ancho de la pantalla. | var=screen.width |
| colorDepth | Bits por pixel, los colores en pantalla. | var=screen.colorDepth |

EJEMPLO:

```

<html>
<head>
    <script>
        function resol()
        {
            var ancho=screen.width;
            var alto=screen.height;

```

```

        if(ancho<1800 && alto<1600)
        {
            alert("Aumentar Resolución a 1800x1600");
            document.write("Aumente la Resolución");
        }
    }
</script>
</head>

<body onLoad=resol();>
</body>
</html>

```

16. MÉTODOS PARA FECHA Y HORA

Métodos que nos van a permitir realizar una serie de operaciones o procedimientos utilizando fechas y horas. Lo primero que tendremos que hacer, construir un objeto `Date`, para que posteriormente podamos utilizar los métodos de fecha y hora.

SINTAXIS DEL OBJETO:

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

| MÉTODOS DE FECHA Y HORA | |
|--|--------------------|
| MÉTODO | DESCRIPCIÓN |
| <code>objeto.toGMTString();</code> | |
| <code>objeto.getDate();</code> | |
| <code>objeto.getMonth()+1;</code> | |
| <code>objeto.getYear();</code> | |
| <code>objeto.getHours();</code> <code>objeto.getMinutes();</code> | |
| <code>objeto.getSeconds();</code> | |
| Todas estas funciones tienen su pareja que nos permite modificar sus valores. Su sintaxis es la misma, solo que ahora comienzan por <code>set</code> . Por ejemplo <code>setMinutes(minutos)</code> cambiará los minutos de la hora del sistema. | |

EJEMPLO 1:

```
<html>
  <head>
    <script>
      function fecha()
      {
        var obj_fecha=new Date();

        var completo=obj_fecha.toGMTString();

        var hora=obj_fecha.getHours();
        var minuto=obj_fecha.getMinutes();
        var segundo=obj_fecha.getSeconds();

        var dia=obj_fecha.getDate();
        var mes=obj_fecha.getMonth()+1;
        var anis=obj_fecha.getFullYear();

        alert(hora +":" +minuto +":" +segundo);
        alert(dia +"/" +mes +"/" +anis);
        alert(completo);
      }
    </script>
  </head>

  <body onLoad=fecha();>
  </body>
</html>
```

EJEMPLO 2: Creación de un reloj digital.

```
<html>
  <head>
    <script>
      setTimeout("reloj()",100);
      function reloj()
      {
        var tiempo=new Date();
        var hora=tiempo.getHours();
        var minuto=tiempo.getMinutes();
        var segundo=tiempo.getSeconds();
        var
textohora=hora+":"+minuto+":"+segundo;
        caja.value=textohora;
        setTimeout("reloj()",500);
      }
    </script>
  </head>
```

```

<body>
  <input type="text" name="caja" size="10">
</body>
</html>

```

17. MÉTODOS MATEMÁTICAS

Contiene los métodos para realizar cualquier tipo de operación matemática compleja. Solo necesitamos llamar al método deseado utilizando el objeto Math (ya está creado). Los métodos mas útiles son:

| <i>MÉTODOS MATEMÁTICOS</i> | |
|--|---|
| <i>MÉTODO</i> | <i>DESCRIPCIÓN</i> |
| <code>var=Math.sin(valor);</code> | Seno de un ángulo. Expresado en radianes. |
| <code>var=Math.cos(valor);</code> | Coseno de un ángulo. Expresado en radianes. |
| <code>var=Math.tan(valor);</code> | Tangente de un ángulo. Expresado en radianes. |
| <code>asin(),acos(),atan()</code> | Igual a las anteriores pero devuelve arcos. |
| <code>var=Math.abs(valor);</code> | Valor absoluto (su positivo). |
| <code>var=Math.log(valor);</code> | Logaritmo decimal de valor. |
| <code>var=Math.max(val1,val2);</code> | Devuelve el mayor de los dos. |
| <code>var=Math.min(val1,val2);</code> | Devuelve el menor de los dos. |
| <code>var=Math.pow(base,expon);</code> | Potencia de base |
| <code>var=Math.sqrt(valor);</code> | Raíz cuadrada del valor. |
| <code>var=Math.round(valor);</code> | Redondea un valor con decimales. |
| <code>var=Math.random()*interv;</code> | Número aleatorio entre 0 y el intervalo. |

EJEMPLO 1:

```

<html>
<head>
  <script>
    function opera(valor)
    {
      var resultado;
      var num1=parseInt(cnum1.value);

```

```

        var num2=parseInt(cnum2.value);

        switch(valor)
        {
            case 1:
                resultado=Math.sqrt(Math.abs(num1));
                break;
            case 2:
                resultado=Math.pow(num1,num2);
                break;
            case 3:
                resultado=Math.max(num1,num2);
                break;
        }
        resul.value=resultado;
    }
</script>
</head>

<body>
    NUM1:
    <input type="text" name="cnum1" size="5"
value="0">
    <br>
    NUM2:
    <input type="text" name="cnum2" size="5"
value="0">
    <br>
    <br>
    <input type="button" value="Raiz"
onClick=opera(1);>
    <input type="button" value="Potencia"
onClick=opera(2);>
    <input type="button" value="Máximo"
onClick=opera(3);>
    <br>
    <br>
    Resultado:<input type="text" name="resul"
size="17">
</body>
</html>

```

EJEMPLO 2: Creamos números aleatorios entre 1 y 10 para luego intentar acertarlo.

```

<html>
<head>
<script>
    var secreto;

    function genera()

```

```

    {
        secreto=Math.random()*10;
        secreto=Math.round(secreto);
        if(secreto==0)
            secreto=10;
    }

    function juego()
    {
        var numero=parseInt(caja.value);
        if(numero==secreto)
        {
            alert("Muy bien.\n A por otro");
            genera();
        }
        else
        {
            if(numero<secreto)
                alert("Es mayor");
            else
                alert("es menor");
        }
        caja.focus();
        caja.value="";
    }
</script>
</head>

<body onLoad=genera();>
    NUMERO:<input type="text" name="caja"
size="3">
    <input type="button" value="Compara"
onClick=juego();>
</body>
</html>

```

18. MÉTODOS PARA CADENAS

Métodos destinados a realizar operaciones con cadenas de texto. Al igual que las funciones matemáticas vamos a necesitar un objeto. En este caso el objeto puede ser una variable normal y corriente que va a contener texto o también podríamos construir un objeto *String*. En ambos casos se trabajará del mismo modo.

SINTAXIS DEL OBJETO String:

```
var nombre_objeto=new String();
```

| <i>MÉTODO</i> | <i>DESCRIPCIÓN</i> |
|-----------------------------------|---|
| objeto/var.length; | Devuelve la longitud de la cadena. |
| objeto/var.charAt(indice); | Devuelve la letra que este en la posición del índice. |
| objeto/var.substring(ind1, ind2); | Devuelve el texto comprendido entre los índices. |
| objeto/var.indexOf(letra); | Devuelve el índice de la letra buscada. |
| objeto/var.replace(letr1, letr2); | Reemplaza letr1 por letr2. |
| objeto/var.toLowerCase(); | Transforma en minúsculas el texto del objeto. |
| objeto/var.toUpperCase(); | Transforma en mayúsculas el texto del objeto. |

EJEMPLO: Pedimos una contraseña de mínimo 4 letras, la pasamos a mayúsculas y mostramos la primera letra. Si la contraseña no llega a 4 letras le avisamos del mínimo de letras y volvemos a pedir.

```

<html>
  <head>
    <script>
      function pasa()
      {
        var contra=T.value;
        var nletras=contra.length;
        if(nletras<4)
        {
          alert("Mínimo 4 letras");
          T.value="";
          nletras=0;
        }
        if(nletras!=0)
        {
          contra=contra.toUpperCase();
          alert("La primera es:
"+contra.charAt(0));
        }
      }
    </script>
  </head>
  <body>
    <input type="password" name="T" size="9"
onBlur=pasa();>
  </body>
</html>

```

INDICE

| | |
|--|----|
| 1. INTRODUCCIÓN | 2 |
| 2. NORMAS DEL CODIGO EN JAVASCRIPT | 2 |
| 3. USO DE FUNCIONES | 3 |
| 4. LA VENTANA "ALERT" | 3 |
| 5. PRIMER PROGRAMA | 4 |
| 6. EVENTOS | 5 |
| 7. VARIABLES Y CONSTANTES | 7 |
| VARIABLES: | 7 |
| CONSTANTES ("literales") | 9 |
| 8. OPERADORES | 10 |
| ASIGNACION (=) | 10 |
| ARITMÉTICOS | 10 |
| LÓGICOS Y RELACIONALES | 11 |
| 9. INTRODUCCIÓN DE DATOS | 11 |
| VENTANA PROMPT: | 12 |
| CAJA DE TEXTO: | 12 |
| 10. SENTENCIAS DE CONTROL | 13 |
| IF-ELSE: | 13 |
| SWITCH: | 15 |
| WHILE: | 17 |
| DO..WHILE: | 17 |
| FOR: | 17 |
| 11. PASO DE PARÁMETROS A FUNCIONES | 21 |
| 12. MATRICES (<i>Arrays</i>) | 23 |
| 13. CONTROLES | 24 |
| 14. ANIMACIONES | 28 |
| 15. OBJETOS PREDEFINIDOS | 33 |
| WINDOW: | 34 |
| DOCUMENT: | 37 |
| HISTORY: | 39 |
| NAVIGATOR: | 40 |
| SCREEN: | 41 |
| 16. MÉTODOS PARA FECHA Y HORA | 42 |
| 17. MÉTODOS MATEMÁTICAS | 44 |
| 18. MÉTODOS PARA CADENAS | 46 |