Estructura repetitiva (while)

Hasta ahora hemos empleado estructuras SECUENCIALES y CONDICIONALES. Existe otro tipo de estructuras tan importantes como las anteriores que son las estructuras REPETITIVAS.
Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:
- La o las sentencias que se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

Funcionamiento del while: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos entre las llaves que le siguen al while. Si la condición sea Falsa continúa con la instrucción siguiente al bloque de llaves. El bloque se repite MIENTRAS la condición sea Verdadera.

Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, nunca finalizará el programa.

Ejemplo: Realizar un programa que imprima en pantalla los números del 1 al 100. Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente. Pero esta solución es muy larga. La mejor forma de resolver este problema es emplear una estructura repetitiva:

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var x;
  x=1;
  while (x<=100)   {
    document.write(x);
    document.write('<br>');
    x=x+1;
  }
</script>
</body>
</html>
```

Para que se impriman los números, uno en cada línea, agregamos la marca HTML de
.
Es muy importante analizar este programa:
La primera operación inicializa la variable x en 1, seguidamente comienza la estructura repetitiva while y disponemos la siguiente condición (x <= 100), se lee MIENTRAS la variable x sea menor o igual a 100.
Al ejecutarse la condición, retorna VERDADERO, porque el contenido de x (1) es menor o igual a 100.
Al ser la condición verdadera se ejecuta el bloque de instrucciones que contiene la estructura while. El bloque de instrucciones contiene dos salidas al documento y una operación. Se imprime el contenido de x y seguidamente se incrementa la variable x en uno.

La operación x = x + 1 se lee como "en la variable x se guarda el contenido de x más 1". Es decir, si x contiene 1 luego de ejecutarse esta operación se almacenará en x un 2.

Al finalizar el bloque de instrucciones que contiene la estructura repetitiva, se verifica nuevamente la condición de la estructura repetitiva y se repite el proceso explicado anteriormente.
Mientras la condición retorne verdadero, se ejecuta el bloque de instrucciones; al retornar falso la verificación de la condición, se sale de la estructura repetitiva y continúa el algoritmo, en este caso, finaliza el programa.

Lo más difícil es la definición de la condición de la estructura while y qué bloque de instrucciones se va a repetir. Observar que si, por ejemplo, disponemos la condición x >=100 (si x es mayor o igual a 100) no provoca ningún error sintáctico, pero estamos en presencia de un error lógico porque al evaluarse por primera vez la condición retorna falso y no se ejecuta el bloque de instrucciones que queríamos repetir 100 veces.

No existe una RECETA para definir una condición de una estructura repetitiva, sino que se logra con una práctica continua, solucionando problemas. Una vez planteado el programa debemos verificar si el mismo es una solución válida al problema (en este caso se deben imprimir los números del 1 al 100 en la página), para ello podemos hacer un

seguimiento del flujo del diagrama y los valores que toman las variables a lo largo de la ejecución:

x	
1	
2	
3	
4	
.	
.	
100	
101	Cuando x vale 101 la condición de la estructura repetitiva retorna falso, en este caso finaliza el diagrama.

La variable x recibe el nombre de **CONTADOR**. Un contador es un tipo especial de variable que se incrementa o decrementa con valores constantes durante la ejecución del programa. El contador x nos indica en cada momento la cantidad de valores impresos en la página.
Importante: Podemos observar que el bloque repetitivo puede no ejecutarse si la condición retorna falso la primera vez.
La variable x debe estar **inicializada** con algún valor antes que se ejecute la operación x = x + 1.

Probemos algunas modificaciones de este programa y veamos qué cambios se deberían hacer para:
1 - Imprimir los números del 1 al 500.
2 - Imprimir los números del 50 al 100.
3 - Imprimir los números del -50 al 0.
4 - Imprimir los números del 2 al 100 pero de 2 en 2 (2,4,6,8 ....100).

25. Realizar un programa que imprima 25 términos de la serie 11 - 22 - 33 - 44, etc. (No se ingresan valores por teclado).

Problema 25

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

<script type="text/javascript">
  var serie;
  serie=11;
  var x;
  x=1;
  while (x<=25)
  {
    document.write(serie);
    document.write('<br>');
    x=x+1;
    serie=serie+11;
  }
</script>

</body>
</html>
```

26. Mostrar los múltiplos de 8 hasta el valor 500. Debe aparecer en pantalla 8 -16 -24, etc.

Problema 26.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var multiplo8;
  multiplo8=8;
  while (multiplo8<=500)
  {
    document.write(multiplo8);
    document.write('<br>');
    multiplo8=multiplo8+8;
  }
</script>
</body>
</html>
```

Concepto de acumulador

Explicaremos el concepto de un acumulador con un ejemplo: desarrollar un programa que permita la carga de 5 valores por teclado y nos muestre posteriormente la suma.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var x=1;
  var suma=0;
  var valor;
  while (x<=5)
  {
    valor=prompt('Ingrese valor:','');
    valor=parseInt(valor);
    suma=suma+valor;
    x=x+1;
  }
  document.write("La suma de los valores es "+suma+"<br>");
</script>
</body>
</html>
```

En este problema, a semejanza de los anteriores, llevamos un CONTADOR llamado x que nos sirve para contar las vueltas que debe repetir el while.
También aparece el concepto de **ACUMULADOR** (un acumulador es un tipo especial de variable que se incrementa o decrementa con valores variables durante la ejecución del programa).
Hemos dado el nombre de suma a nuestro acumulador. Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementa con el contenido ingresado en la variable valor.

La prueba del diagrama se realiza dándole valores a las variables:

valor	suma	x
0	0	

(Antes de entrar a la estructura repetitiva estos son los valores).

5	5	1
16	21	2
7	28	3
10	38	4
2	40	5

Este es un seguimiento del programa planteado. Los números que toma la variable valor dependerá de qué cifras cargue el operador durante la ejecución del programa.
Hay que tener en cuenta que cuando en la variable valor se carga el primer valor (en este ejemplo es el valor 5), al cargarse el segundo valor (16), el valor anterior 5 se pierde, por ello la necesidad de ir almacenando en la variable suma el valor acumulado de los valores ingresados.

27. Escribir un programa que lea 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.

Problema 27.

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

<script type="text/javascript">
  var x=0;
  var cant1=0;
  var cant2=0;
  var nota;
  while (x<10) {
    nota=prompt('Ingrese nota','');
    if (nota>=7) {
      cant1=cant1+1;
    } else {
      cant2=cant2+1;
    }
    x=x+1;
  }
  document.write('Cantidad de alumnos con notas mayores o iguales a 7:'+cant1);
  document.write('<br>');
  document.write('Cantidad de alumnos con notas menores a 7:'+cant2);
</script>

</body>
</html>
```

28. Se ingresan un conjunto de 5 alturas de personas por teclado. Mostrar la altura promedio de las personas.

```
Problema 28.

<html>
<head>
</head>
<body>

<script type="text/javascript">
  var x=0;
  var suma=0;
  var altura;
  while (x<5) {
    altura=prompt('Ingrese la altura en centímetros(Ej. 175)','');
    altura=parseInt(altura);
    suma=suma+altura;
    x=x+1;
  }
  var promedio=suma/5;
  document.write('La altura promedio de las cinco personas es:'+promedio);
</script>

</body>
</html>
```

29. En una empresa trabajan 5 empleados cuyos sueldos oscilan entre 100€ y 500€. Realizar un programa que lea los sueldos que cobra cada empleado e informe cuántos empleados cobran menos de 300€ y cuántos cobran más de 300€. Además, el programa deberá informar el importe que gasta la empresa en sueldos al personal.

Problema 29.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var cont1=0;
  var cont2=0;
  var total=0;
  var sueldo;
  var x=0;
  while (x<5) {
    sueldo=prompt ('Ingrese el sueldo','');
    sueldo=parseInt(sueldo);
    if (sueldo<=300) {
      cont1=cont1+1;
    } else {
      cont2=cont2+1;
    }
    total=total+sueldo;
    x=x+1;
  }
  document.write('Cantidad de empleados que cobran 300 o menos:'+cont1);
  document.write('<br>');
  document.write('Cantidad de empleados que cobran más de 300:'+cont2);
  document.write('<br>');
  document.write('Gastos en sueldos en la empresa:'+total);
</script>
</body>
</html>
```

30. Realizar un programa que permita cargar dos listas de 3 valores cada una. Informar con un mensaje cuál de las dos listas tiene un valor acumulado mayor (mensajes 'Lista 1 mayor', 'Lista 2 mayor', 'Listas iguales') Tener en cuenta que puede haber dos o más estructuras repetitivas en un algoritmo.

Problema 30.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var total1=0;
  var x=0;
  var nro;
  while(x<3){
    nro=prompt('Ingrese valor de la primer lista:','');
    nro=parseInt(nro);
    total1=total1+nro;
    x=x+1;
  }
  var total2=0;
  x=0;
  while(x<3){
    nro=prompt('Ingrese valor de la segunda lista:','');
    nro=parseInt(nro);
    total2=total2+nro;
    x=x+1;
  }
  if (total1>total2) {
    document.write('Tiene mayor valor la lista1');
  } else {
    if (total1<total2) {
      document.write('Tiene mayor valor la lista2');
    } else {
      document.write('Tienen el mismo valor acumulado las dos listas');
    }
  }
</script>
</body>
</html>
```

31. Desarrollar un programa que permita cargar 5 números enteros y luego nos informe cuántos valores fueron pares y cuántos impares.
Emplear el operador "%" en la condición de la estructura condicional.

```
Problema 31.

<html>
<head>
</head>
<body>

<script type="text/javascript">
  var cantpares=0;
  var cantimpares=0;
  var x=0;
  var valor;
  while (x<5){
    valor=prompt('Ingrese un valor','');
    valor=parseInt(valor);
    if (valor%2==0)
    {
      cantpares=cantpares+1;
    }
    else
    {
      cantimpares=cantimpares+1;
    }
    x=x+1;
  }
  document.write('Cantidad de valores pares ingresados:'+cantpares);
  document.write('<br>');
  document.write('Cantidad de valores impares ingresados:'+cantimpares);
</script>

</body>
</html>
```

Estructura repetitiva (do - while)

La sentencia do/while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while que puede no ejecutar el bloque.
Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.
La condición de la estructura está abajo del bloque a repetir, a diferencia del while que está en la parte superior.
Finaliza la ejecución del bloque repetitivo cuando la condición retorna falso, es decir igual que el while.

Problema: Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuántos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var valor;
  do {
    valor=prompt('Ingrese un valor entre 0 y 999:','');
    valor=parseInt(valor);
    document.write('El valor '+valor+' tiene ');
    if (valor<10)
    {
      document.write('Tiene 1 digitos');
    }
    else
    {
      if (valor<100)
      {
        document.write('Tiene 2 digitos');
      }
      else
      {
        document.write('Tiene 3 digitos');
      }
    }
   document.write('<br>');
  } while(valor!=0);
</script>
</body>
</html>
```

En este problema por lo menos se carga un valor. Si se carga un valor menor a 10 se trata de un número de una cifra, si es mayor a 10 pero menor a 100 se trata de un valor de dos dígitos, en caso contrario se trata de un valor de tres dígitos. Este bloque se repite mientras se ingresa en la variable 'valor' un número distinto a 0.

32. Realizar un programa que acumule (sume) valores ingresados por teclado hasta ingresar el 9999 (no sumar dicho valor, solamente indica que ha finalizado la carga). Imprimir el valor acumulado e informar si dicho valor es cero, mayor a cero o menor a cero.

Problema 32.

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

<script type="text/javascript">
  var valor;
  var suma=0;
  do {
    valor=prompt('Ingrese un valor (9999 para finalizar)','');
    valor=parseInt(valor);
    if (valor!=9999)
    {
      suma=suma+valor;
    }
  } while(valor!=9999);
  document.write('Valor acumulado total:'+suma);
  document.write('<br>');
  if (suma>0)
  {
    document.write('El valor acumulado es mayor a cero');
  }
  else
  {
    if (suma==0)
    {
      document.write('El valor acumulado es cero');
    }
    else
    {
      document.write('El valor acumulado es menor a cero');
    }
  }
</script>

</body>
</html>
```

33. En un banco se procesan datos de las cuentas corrientes de sus clientes. De cada cuenta corriente se conoce: número de cuenta, nombre del cliente y saldo actual. El ingreso de datos debe finalizar al ingresar un valor negativo en el número de cuenta.
Se pide confeccionar un programa que lea los datos de las cuentas corrientes e informe:

a) De cada cuenta: número de cuenta, nombre del cliente y estado de la cuenta según su saldo, sabiendo que:

Estado de la cuenta 'Acreedor' si el saldo es >0.
'Deudor' si el saldo es <0.
'Nulo' si el saldo es =0.

b) La suma total de los saldos acreedores.

Problema 33.

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

<script type="text/javascript">
  var nrocuenta;
  var nombre;
  var saldo=0;
  var saldoacre=0;
  do {
    nrocuenta=prompt('Ingrese nro de cuenta:','');
    nrocuenta=parseInt(nrocuenta);
    if (nrocuenta>=0) {
      nombre=prompt('Nombre del cliente:');
      saldo=prompt('Saldo actual:','');
      saldo=parseInt(saldo);
      if (saldo>0)  {
        saldoacre=saldoacre+saldo;
        document.write(nombre+' tiene saldo acreedor<br>');
      } else {
        if (saldo<0) {
          document.write(nombre+' tiene saldo deudor<br>');
        } else {
          document.write(nombre+' tiene saldo nulo<br>');
        }
      }
    }
  }while(nrocuenta>0);
  document.write('Suma total de saldos acreedores:'+saldoacre);
</script>

</body>
</html>
```

34. Se realizó un censo provincial y se desea procesar la información obtenida en dicho censo. De cada una de las personas censadas se tiene la siguiente información: número de documento, edad y sexo ('femenino' o 'masculino')
Se pide confeccionar un programa que lea los datos de cada persona censada (para finalizar ingresar el valor cero en el número de documento) e informar:

a) Cantidad total de personas censadas.
b) Cantidad de varones.
c) Cantidad de mujeres.
d) Cantidad de varones cuya edad varía entre 16 y 65 años.

Problema 34.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var documento;
  var edad;
  var totalpersonas=0;
  var cantvarones=0;
  var cantmujeres=0;
  var cantvaronesgrandes=0;
  do {
    documento=prompt('Ingrese nro de documento:','');
    documento=parseInt(documento);
    if (documento>0) {
      edad=prompt('Ingrese la edad:','');
      edad=parseInt(edad);
      sexo=prompt('Ingrese el sexo (masculino/femenino):','');
      if (sexo=='masculino') {
        cantvarones=cantvarones+1;
        if (edad>=16 && edad<=65) {
          cantvaronesgrandes=cantvaronesgrandes+1;
        }
      }
      if (sexo=='femenino') {
        cantmujeres=cantmujeres+1;
      }
      totalpersonas=totalpersonas+1;
    }
  }while(documento!=0);
  document.write('Total de personas censadas: '+totalpersonas+'<br>');
  document.write('Cantidad de varones: '+cantvarones+'<br>');
  document.write('Cantidad de mujeres: '+cantmujeres+'<br>');
  document.write('Cantidad de varones entre 16 y 65 años: '+cantvaronesgrandes+'<br>');
</script>
</body>
</html>
```

Estructura repetitiva (for)

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while. Pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones.
Esta estructura se emplea en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita.

Por último, hay que decir que la ejecución de la sentencia break dentro de cualquier parte del bucle provoca la salida inmediata del mismo.

Sintaxis:

```
for (<Inicialización> ; <Condición> ; <Incremento o Decremento>)
{
  <Instrucciones>
}
```

Esta estructura repetitiva tiene tres argumentos: variable de inicialización, condición y variable de incremento o decremento.

Funcionamiento:

- Primero se ejecuta por única vez el primer argumento. Por lo general se inicializa una variable.
- El segundo paso es evaluar la (Condición), en caso de ser verdadera se ejecuta el bloque, en caso contrario continúa el programa.
- El tercer paso es la ejecución de las instrucciones.
- El cuarto paso es ejecutar el tercer argumento (Incremento o Decremento).
- Luego se repiten sucesivamente del Segundo al Cuarto Paso.

Este tipo de estructura repetitiva se utiliza generalmente cuando sabemos la cantidad de veces que deseamos que se repita el bloque.

Ejemplo: Mostrar por pantalla los números del 1 al 10.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var f;
  for(f=1; f<=10; f++) {
    document.write(f+" ");
  }
</script>
</body>
</html>
```

Inicialmente f se la inicializa con 1. Como la condición se verifica como verdadera se ejecuta el bloque del for (en este caso mostramos el contenido de la variable f y un espacio en blanco). Luego de ejecutar el bloque pasa al tercer argumento del for (en este caso con el operador ++ se incrementa en uno el contenido de la variable f, existe otro operador -- que decrementa en uno una variable), hubiera sido lo mismo poner f=f+1 pero este otro operador matemático nos simplifica las cosas.

Importante: Tener en cuenta que no lleva punto y coma al final de los tres argumentos del for. El disponer un punto y coma provoca un error lógico y no sintáctico, por lo que el navegador no avisará.

35. Escribir un programa que pida ingresar coordenadas (x,y) que representan puntos en el plano. Informar cuántos puntos se han ingresado en el primer, segundo, tercer y cuarto cuadrante. Al comenzar el programa se pide que se ingrese la cantidad de puntos a procesar.

Problema 35.

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

<script type="text/javascript">
/*Escribir un programa que pida ingresar coordenadas (x,y) que representan puntos en el plano.
Informar cuántos puntos se han ingresado en el primer, segundo, tercer y cuarto cuadrante.
Al comenzar el programa se pide que se ingrese la cantidad de puntos a procesar.*/

  var cuad1=0;
  var cuad2=0;
  var cuad3=0;
  var cuad4=0;
  var x, y, f, cant;
  cant=prompt('Cuantos puntos procesará:','');
  cant=parseInt(cant);
  for(f=1;f<=cant;f++)  {
    x=prompt('Ingrese coordenada x:','');
    x=parseInt(x);
    y=prompt('Ingrese coordenada y:','');
    y=parseInt(y);
    if (x>0 && y>0)    {
      cuad1++;
    }     else    {
      if (x<0 && y>0)      {
        cuad2++;
      }     else       {
        if (x<0 && y<0)        {
          cuad3++;
        }         else        {
          if (x>0 && y<0)          {
            cuad4++;
          }
        }
      }
    }
  }
  document.write('Cantidad de puntos ingresados en el primer cuadrante:'+cuad1+'<br>');
  document.write('Cantidad de puntos ingresados en el segundo cuadrante:'+cuad2+'<br>');
  document.write('Cantidad de puntos ingresados en el tercer cuadrante:'+cuad3+'<br>');
  document.write('Cantidad de puntos ingresados en el cuarto cuadrante:'+cuad4+'<br>');
</script>

</body>
</html>
```

36. Se realiza la carga de 10 valores enteros por teclado. Se desea conocer:
a) La cantidad de valores negativos ingresados.
b) La cantidad de valores positivos ingresados.
c) La cantidad de múltiplos de 15.
d) El valor acumulado de los números ingresados que son pares.

```
Problema 36.

<html>
<head>
</head>
<body>
<script type="text/javascript">
/*Se realiza la carga de 10 valores enteros por teclado. Se desea conocer:
  a)        La cantidad de valores negativos ingresados.
  b)        La cantidad de valores positivos ingresados.
  c)        La cantidad de múltiplos de 15.
  d)        El valor acumulado de los números ingresados que son pares.*/

  var cantnegativos=0;
  var cantpositivos=0;
  var mult15=0;
  var sumapares=0;
  var f;
  var valor;
  for(f=1;f<=10;f++)  {
    valor=prompt('Ingrese un valor:','');
    valor=parseInt(valor);
    if (valor<0)   {
      cantnegativos++;
    } else  {
      if (valor>0)  {
        cantpositivos++;
      }
    }
    if (valor%15==0 && valor!=0)   {
      mult15++;
    }
    if (valor%2==0)   {
      sumapares=sumapares+valor;
    }
  }
  document.write('Cantidad de valores negativos:'+cantnegativos+'<br>');
  document.write('Cantidad de valores positivos:'+cantpositivos+'<br>');
  document.write('Cantidad de múltiplos de 15:'+mult15+'<br>');
  document.write('Suma de los valores pares ingresados:'+sumapares+'<br>');
</script>

</body>
</html>
```

Funciones

En programación es muy frecuente que un determinado procedimiento de cálculo definido por un grupo de sentencias tenga que repetirse varias veces, ya sea en un mismo programa o en otros programas, lo cual implica que se tenga que escribir tantos grupos de aquellas sentencias como veces aparezca dicho proceso.
La herramienta más potente con que se cuenta para facilitar, reducir y dividir el trabajo en programación, es escribir aquellos grupos de sentencias una sola y única vez bajo la forma de una FUNCION.
Un programa es una cosa compleja de realizar y por lo tanto es importante que esté bien ESTRUCTURADO y también que sea inteligible para las personas. Si un grupo de sentencias realiza una tarea bien definida, entonces puede estar justificado el aislar estas sentencias formando una función, aunque resulte que sólo se le llame o use una vez.
Hasta ahora hemos visto cómo resolver un problema planteando un único algoritmo.
Con funciones podemos segmentar un programa en varias partes.
Frente a un problema, planteamos un algoritmo, éste puede constar de pequeños algoritmos.
Una función es un conjunto de instrucciones que resuelven una parte del problema y que puede ser utilizado (llamado) desde diferentes partes de un programa.
Consta de un nombre y parámetros. Con el nombre llamamos a la función, es decir, hacemos referencia a la misma. Los parámetros son valores que se envían y son indispensables para la resolución del mismo. La función realizará alguna operación con los parámetros que le enviamos. Podemos cargar una variable, consultarla, modificarla, imprimirla, etc.

Incluso los programas más sencillos tienen la necesidad de fragmentarse.
Las funciones son los únicos tipos de subprogramas que acepta JavaScript.
Tienen la siguiente estructura:

```
function <nombre de función>(argumento1, argumento2, ..., argumento n)
{
  <código de la función>
}
```

Debemos buscar un nombre de función que nos indique cuál es su objetivo (Si la función recibe un string y lo centra, tal vez deberíamos llamarla centrarTitulo). Veremos que una función puede variar bastante en su estructura, puede tener o no parámetros, retornar un valor, etc.

Ejemplo: Mostrar un mensaje que se repita 3 veces en la página con el siguiente texto:

'Cuidado'
'Ingrese su documento correctamente'

'Cuidado'
'Ingrese su documento correctamente'

'Cuidado'
'Ingrese su documento correctamente'

La solución sin emplear funciones es:

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

<script type="text/javascript">
  document.write("Cuidado<br>");
  document.write("Ingrese su documento correctamente<br>");
  document.write("Cuidado<br>");
  document.write("Ingrese su documento correctamente<br>");
  document.write("Cuidado<br>");
  document.write("Ingrese su documento correctamente<br>");
</script>

</body>
</html>
```

Empleando una función:

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

<script type="text/javascript">
  function mostrarMensaje()
  {
    document.write("Cuidado<br>");
    document.write("Ingrese su documento correctamente<br>");
  }

  mostrarMensaje();
  mostrarMensaje();
  mostrarMensaje();
</script>

</body>
</html>
```

Recordemos que JavaScript es sensible a mayúsculas y minúsculas. Si fijamos como nombre a la función mostrarTitulo (es decir la segunda palabra con mayúscula) debemos respetar este nombre cuando la llamemos a dicha función.
Es importante notar que para que una función se ejecute debemos llamarla desde fuera por su nombre (en este ejemplo: mostrarMensaje()).
Cada vez que se llama una función se ejecutan todas las líneas contenidas en la misma.
Si no se llama a la función, las instrucciones de la misma nunca se ejecutarán.
A una función la podemos llamar tantas veces como necesitemos.
Las funciones nos ahorran escribir código que se repite con frecuencia y permite que nuestro programa sea más entendible.

Funciones con parámetros

Explicaremos con un ejemplo, una función que tiene datos de entrada.
Ejemplo: Confeccionar una función que reciba dos números y muestre en la página los valores comprendidos entre ellos de uno en uno. Cargar por teclado esos dos valores.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">

  function mostrarComprendidos(x1,x2)  {
    var inicio;
    for(inicio=x1;inicio<=x2;inicio++)  {
      document.write(inicio+' ');
    }
  }

  var valor1,valor2;
  valor1=prompt('Ingrese valor inferior:','');
  valor1=parseInt(valor1);
  valor2=prompt('Ingrese valor superior:','');
  valor2=parseInt(valor2);
  mostrarComprendidos(valor1,valor2);

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

El programa de JavaScript empieza a ejecutarse donde definimos las variables valor1 y valor2 y no donde se define la función. Después de cargar los dos valores por teclado se llama a la función mostrarComprendidos y le enviamos las variables valor1 y valor2. Los parámetros x1 y x2 reciben los contenidos de las variables valor1 y valor2.
Es importante notar que a la función la podemos llamar la cantidad de veces que la necesitemos. Los nombres de los parámetros, en este caso se llaman x1 y x2, no necesariamente se deben llamar igual que las variables que le pasamos cuando la llamamos a la función, en este caso le pasamos los valores valor1 y valor2.

Funciones que retornan un valor

Son comunes los casos donde una función, luego de hacer un proceso, retorne un valor.
Por ejemplo: Confeccionar una función que reciba un valor entero comprendido entre 1 y 5. Luego retornar en castellano el valor recibido.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">

  function convertirCastellano(x)  {
    if (x==1)
      return "uno";
    else if (x==2)
      return "dos";
    else if (x==3)
      return "tres";
    else if (x==4)
      return "cuatro";
    else if (x==5)
      return "cinco";
    else
      return "valor incorrecto";
  }
  var valor, r;
  valor=prompt("Ingrese un valor entre 1 y 5","");
  valor=parseInt(valor);
  r=convertirCastellano(valor);
  document.write(r);
</script>
</body>
</html>
```

Podemos ver que el valor retornado por una función lo indicamos por medio de la palabra clave **return**. Cuando se llama a la función, debemos asignar el nombre de la función a una variable, ya que la misma retorna un valor. Una función puede tener varios parámetros, pero sólo puede retornar un único valor.
La estructura condicional if de este ejemplo puede ser remplazada por la instrucción switch, la función queda codificada de la siguiente manera:

```
function convertirCastellano(x)
{
  switch (x)
  {
    case 1:return "uno";
    case 2:return "dos";
    case 3:return "tres";
    case 4:return "cuatro";
    case 5:return "cinco";
    default:return "valor incorrecto";
  }
}
```

Esta es una forma más elegante que una serie de if anidados. La instrucción switch analiza el contenido de la variable x con respecto al valor de cada caso. En la situación de ser igual, ejecuta el bloque seguido de los 2 puntos hasta que encuentra la instrucción return o break.

37. Confeccionar una función que reciba una fecha con el formato de día, mes y año y retorne un string con un formato similar a: "Hoy es 10 de junio de 2013".

```
<html>
<head>
</head>
<body>
<script type="text/javascript">


 function formatearFecha(dia,mes,año) {
   var s='Hoy es '+dia+' de ';
   switch (mes) {
   case 1:s=s+'enero ';
          break;
   case 2:s=s+'febrero ';
          break;
   case 3:s=s+'marzo ';
          break;
   case 4:s=s+'abril ';
          break;
   case 5:s=s+'mayo ';
          break;
   case 6:s=s+'junio ';
          break;
   case 7:s=s+'julio ';
          break;
   case 8:s=s+'agosto ';
          break;
   case 9:s=s+'septiembre ';
          break;
   case 10:s=s+'octubre ';
          break;
   case 11:s=s+'noviembre ';
          break;
   case 12:s=s+'diciembre ';
          break;
   } //fin del switch
   s=s+'de '+año;
   return s;
 }

 document.write(formatearFecha(11,6,2013));

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

Cuando se llama a la función directamente, al valor devuelto se lo enviamos a la función write del objeto document. Esto último lo podemos hacer en dos pasos:

```
var fec= formatearFecha(11,6,2013);
document.write(fec);
```

Guardamos en la variable 'fec' el string devuelto por la función.

Funciones: variables locales y variables globales

Las variables locales se definen dentro de una función y solo se tiene acceso dentro de la misma función. Una variable local se la define antecediendo la palabra clave var al nombre de la variable.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">

  function imprimir(){
    var x=10;
    document.write(x);
  }

  imprimir();

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

Luego decimos que x es una variable local que solo se la puede acceder dentro de la función imprimir. Si intentamos acceder a la variable x fuera de la función se produce un error en tiempo de ejecución:

```
<script type="text/javascript">

  function imprimir(){
    var x=10;
    document.write(x);
  }

  imprimir();
  document.write(x);  //error en tiempo de ejecución

</script>
```

Las variables globales son las que definimos fuera de cualquier función. Una variable global tenemos acceso fuera y dentro de las funciones:

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

<script type="text/javascript">

  function imprimir()
  {
    document.write(global1+'<br>');  // muestra un 100
    global1++;
  }

  var global1=100;
  imprimir();
  document.write(global1);  // muestra un 101

</script>

</body>
</html>
```

Como podemos observar el programa anterior hemos definido la variable global1 e inicializado con el valor 100. Luego dentro de la función podemos acceder para imprimirla y modificarla. Podemos ver luego si la accedemos fuera de la función su valor fue modificado.

Todas las variables globales se convierten en atributos del objeto window, luego podemos acceder a la variable por su nombre o antecediendo el nombre del objeto:

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

<script type="text/javascript">

  var global1=100;
  document.write(global1+'<br>');        //100
  document.write(window.global1+'<br>'); //100

</script>

</body>
</html>
```

Cuidado

Cuando definimos variables locales dentro de una función no tenemos que olvidarnos de anteceder al nombre de la variable la palabra var, en caso de olvidarnos se creará automáticamente una variable global con dicho nombre:

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

<script type="text/javascript">

 function imprimir()
 {
   x=100;
   document.write(x+'<br>');
 }

 imprimir();
 document.write(x+'<br>');

</script>

</body>
</html>
```

En el ejemplo superior en la función imprimir se está creando una variable global llamada x. Como podemos ver luego fuera de la función cuando imprimimos x vemos que no produce error y muestra el valor 100. Con solo agregar la palabra clave var vemos que nos muestra un mensaje de error al tratar de acceder a una variable inexistente:

```
 function imprimir()
 {
   var x=100;
   document.write(x+'<br>');
 }

 imprimir();
 document.write(x+'<br>');
```

El JavaScript moderno introduce una directiva para salvar este y otros problemas muy comunes. Si activamos modo estricto todas las variables en JavaScript deben declarase sino se produce un error. Veamos como introducimos esta directiva de modo estricto:

```
<html>
<head>
</head>
<body>
<script type="text/javascript">

  'use strict';

  function imprimir()
  {
    x=100; //error
    document.write(x+'<br>');
  }

  imprimir();
  document.write(x+'<br>');

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

La directiva de modo estricto emplea las palabras claves *use strict* entre comillas simples o dobles y un punto y coma. Solo los navegadores modernos implementan los análisis de modo estricto.

La razón de disponer la directiva entre comillas y con un punto y coma es para que navegadores antiguos pasen por alto el string (ya que el string no se asigna a nadie el navegador antiguo no lo tiene en cuenta).

Activando el modo estricto luego cuando tratamos de asignar el valor 100 a la variable x se produce un error ya que no se nos permite crear una variable global (esto nos evita problemas de crear variables globales en funciones por error).

Clase Date

JavaScript dispone de varias clases predefinidas para acceder a muchas de las funciones normales de cualquier lenguaje, como puede ser el manejo de vectores o el de fechas.

Esta clase nos permitirá manejar fechas y horas. Se invoca así:

```
//creación de un objeto de la clase Date
fecha = new Date();
```

```
fecha = new Date(año, mes, dia);
fecha = new Date(año, mes, dia, hora, minuto, segundo);
```

Si no utilizamos parámetros, el objeto fecha contendrá la fecha y hora actuales, obtenidas del reloj de nuestro ordenador. En caso contrario hay que tener en cuenta que los meses comienzan por cero. Así, por ejemplo:

```
Navidad17 = new Date(2017, 11, 25)
```

El objeto Date dispone, ***entre otros***, de los siguientes métodos:

getYear() → Obtiene el año de la fecha y devuelve un número de 4 dígitos excepto en el caso en que esté entre 1900 y 1999, que devolverá las dos últimas cifras.
getFullYear() →Realiza la misma función, pero siempre devuelve números con todos sus dígitos.
getMonth() → Obtiene el mes de la fecha.
getDate() → Obtiene el día de la fecha.
getDay() → Devuelve el día de la semana de la fecha en forma de número que va del 0 (domingo) al 6 (sábado)
getHours() → Devuelve la hora actual.
getMinutes() → Devuelve los minutos actuales.
getSeconds() → Devuelve los segundos actuales.

Ejemplo: Mostrar en una página la fecha y la hora actual.

```
<html>
<head>
< script type="text/javascript">
  function mostrarFechaHora() {
    var fecha
    fecha=new Date();
    document.write('Hoy es ');
    document.write(fecha.getDate()+'/');
    document.write((fecha.getMonth()+1)+'/');
    document.write(fecha.getFullYear());
    document.write('<br>');
    document.write('Es la hora ');
    document.write(fecha.getHours()+':');
    document.write(fecha.getMinutes()+':');
    document.write(fecha.getSeconds());
  }

//Llamada a la función
  mostrarFechaHora();
</script>
</head>
<body>
</body >
</html >
```

Clase Array

Un vector es una estructura de datos que permite almacenar un CONJUNTO de datos. Con un único nombre se define un vector y por medio de un subíndice hacemos referencia a cada ***componente*** del mismo.

Ejemplo 1: Crear un vector para almacenar los cinco sueldos de operarios y luego mostrar el total de gastos en sueldos (cada actividad en una función).

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  function cargar(sueldos){
    var f;
    var v;
    for(f=0; f<sueldos.length; f++){
      v=prompt('Ingrese sueldo:','');
      sueldos[f]=parseInt(v);
    }
  }

  function calcularGastos(sueldos) {
    var total=0;
    var f;
    for(f=0; f<sueldos.length; f++){
      total=total+sueldos[f];
    }
    document.write('Listado de sueldos<br>');
    for(f=0; f<sueldos.length; f++){
      document.write(sueldos[f]+'<br>');
    }
    document.write('Total de gastos en sueldos:'+total);
  }

  var sueldos;
  sueldos=new Array(5);
  cargar(sueldos);
  calcularGastos(sueldos);
</script>

</body>
</html>
```

Recordemos que el programa comienza a ejecutarse a partir de las líneas que se encuentran fuera de las funciones:

```
var sueldos;
sueldos=new Array(5);
cargar(sueldos);
calcularGastos(sueldos);
```

Lo primero, definimos una variable y posteriormente creamos un objeto de la clase Array, indicándole que queremos almacenar 5 valores.
Llamamos a la función cargar enviándole el vector. En la función, a través de un ciclo for recorremos las distintas componentes del vector y almacenamos valores enteros que ingresamos por teclado.
Para conocer el tamaño del vector accedemos a la propiedad length de la clase Array.
En la segunda función sumamos todas las componentes del vector, imprimimos en la página los valores y el total de gastos.

Ejemplo 2: Crear un vector con elementos de tipo string. Almacenar los meses de año. En otra función solicitar el ingreso de un número entre 1 y 12. Mostrar a qué mes corresponde y cuántos días tiene dicho mes.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  function mostrarFecha(meses,dias){
    var num;
    num=prompt('Ingrese número de mes:','');
    num=parseInt(num);
    document.write('Corresponde al mes:'+meses[num-1]);
    document.write('<br>');
    document.write('Tiene '+dias[num-1]+' días');
  }
  var meses;
  meses = new Array(12);
  meses[0]='Enero';
  meses[1]='Febrero';
  meses[2]='Marzo';
  meses[3]='Abril';
  meses[4]='Mayo';
  meses[5]='Junio';
  meses[6]='Julio';
  meses[7]='Agosto';
  meses[8]='Septiembre';
  meses[9]='Octubre';
  meses[10]='Noviembre';
  meses[11]='Diciembre';
  var dias;
  dias = new Array(12);
  dias[0]=31;
  dias[1]=28;
  dias[2]=31;
  dias[3]=30;
  dias[4]=31;
  dias[5]=30;
  dias[6]=31;
  dias[7]=31;
  dias[8]=30;
  dias[9]=31;
```

```
  dias[10]=30;
  dias[11]=31;
  mostrarFecha(meses,dias);
</script>
</body>
</html>
```

En este problema definimos dos vectores, uno para almacenar los meses y otro los días. Decimos que se trata de vectores paralelos porque en la componente cero del vector meses almacenamos el string 'Enero' y en el vector dias, la cantidad de días del mes de enero.
Es importante notar que cuando imprimimos, disponemos como subíndice el valor ingresado menos 1, esto debido a que normalmente el operador de nuestro programa carga un valor comprendido entre 1 y 12. Recordar que los vectores comienzan a numerarse a partir de la componente cero.

```
document.write('Corresponde al mes:'+meses[num-1]);
```

Clase Math

Esta clase es un contenedor que tiene diversas constantes (como Math.E y Math.PI) y los siguientes métodos matemáticos:

Método	Descripción	Expresión de ejemplo	Resultado del ejemplo
abs	Valor absoluto	Math.abs(-2)	2
sin, cos, tan	Funciones trigonométricas, reciben el argumento en radianes	Math.cos(Math.PI)	-1
asin, acos, atan	Funciones trigonométricas inversas	Math.asin(1)	1.57
exp, log	Exponenciación y logaritmo, base E	Math.log(Math.E)	1
ceil	Devuelve el entero más pequeño mayor o igual al argumento	Math.ceil(-2.7)	-2
floor	Devuelve el entero más grande menor o igual al argumento	Math.floor(-2.7)	-3
round	Devuelve el entero más cercano o igual al argumento	Math.round(-2.7)	-3
min, max	Devuelve el menor (o mayor) de sus dos argumentos	Math.min(2,4)	2
pow	Exponenciación, siendo el primer argumento la base y el segundo el exponente	Math.pow(2,3)	8
sqrt	Raíz cuadrada	Math.sqrt(25)	5
random	Genera un valor aleatorio comprendido entre 0 y 1.	Math.random()	Ej. 0.7345

Ejemplo: Confeccionar un programa que permita cargar un valor comprendido entre 1 y 10. Luego generar un valor aleatorio entre 1 y 10, mostrar un mensaje con el número sorteado e indicar si ganó o perdió:

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

<script type="text/javascript">
  var selec=prompt('Ingrese un valor entre 1 y 10','');
  selec=parseInt(selec);
  var num=parseInt(Math.random()*10)+1;
  if (num==selec)
    document.write('Ganó el número que se sorteó es el '+ num);
  else
    document.write('Lo siento se sorteó el valor '+num+' y usted eligió el '+selec);
</script>

</body>
</html>
```

Para generar un valor aleatorio comprendido entre 1 y 10 debemos plantear lo siguiente:

```
var num=parseInt(Math.random()*10)+1;
```

Al multiplicar Math.random() por 10, nos genera un valor aleatorio comprendido entre un valor mayor a 0 y menor a 10, luego, con la función parseInt, obtenemos sólo la parte entera. Finalmente sumamos uno.
El valor que cargó el operador se encuentra en:

```
var selec=prompt('Ingrese un valor entre 1 y 10','');
```

Con un simple if validamos si coinciden los valores (el generado y el ingresado por teclado).

Clase String

Un string consiste en uno o más caracteres encerrados entre comillas simples o dobles. JavaScript permite **concatenar** cadenas utilizando el operador **+.** El siguiente fragmento de código concatena tres cadenas para producir su salida:

```
var final='La entrada tiene ' + contador + ' caracteres.';
```

Dos de las cadenas concatenadas son cadenas literales. La del medio es un entero que automáticamente se convierte a cadena y luego se concatena con las otras.

length → Retorna la cantidad de caracteres de un objeto String.

```
var nom='Juan';
document.write(nom.length); //Resultado 4
```

charAt(pos) → Retorna el carácter del índice especificado. Comienzan a numerarse de la posición cero.

```
var nombre='juan';
var caracterPrimero=nombre.charAt(0);
```

substring (posinicial, posfinal) → Retorna un String extraída de otro, desde el carácter 'posinicial' hasta el 'posfinal'-1:

```
cadena3=cadena1.substring(2,5);
```

En este ejemplo, "cadena3" contendrá los caracteres 2, 3, 4 sin incluir el 5 de cadena1 (Cuidado que comienza en cero).

indexOf (subCadena) → Devuelve la posición de la subcadena dentro de la cadena, o -1 en caso de no estar. Tener en cuenta que puede retornar 0 si la subcadena coincide desde el primer carácter.

```
var nombre='Rodriguez Pablo';
var pos=nombre.indexOf('Pablo');
if (pos!=-1)
  document.write ('Está el nombre Pablo en la variable nombre');
```

toUpperCase() → Convierte todos los caracteres del String que invoca el método a mayúsculas:

```
cadena1=cadena1.toUpperCase();
```

Después de esto, cadena1 tiene todos los caracteres convertidos a mayúsculas.

toLowerCase() → Convierte todos los caracteres del String que invoca el método a minúsculas:

```
cadena1=cadena1.toLowerCase();
```

Después de esto, cadena1 tiene todos los caracteres convertidos a minúsculas.

Ejemplo: Cargar un string por teclado y luego llamar a los distintos métodos de la clase String y la propiedad length.

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

<script type="text/javascript">
  var cadena=prompt('Ingrese una cadena:','');
  document.write('La cadena ingresada es:'+cadena);
  document.write('<br>');
  document.write('La cantidad de caracteres son:'+cadena.length);
  document.write('<br>');
  document.write('El primer carácter es:'+cadena.charAt(0));
  document.write('<br>');
  document.write('Los primeros 3 caracteres son:'+cadena.substring(0,3));
  document.write('<br>');
  if (cadena.indexOf('hola')!=-1)
    document.write('Se ingresó la subcadena hola');
  else
    document.write('No se ingresó la subcadena hola');
  document.write('<br>');
  document.write('La cadena convertida a mayúsculas es:'+cadena.toUpperCase());
  document.write('<br>');
  document.write('La cadena convertida a minúsculas es:'+cadena.toLowerCase());
  document.write('<br>');
</script>

</body>
</html>
```

Formularios y Eventos

El uso de Javascript en los formularios HTML se hace fundamentalmente con el objetivo de validar los datos ingresados. Se hace esta actividad en el cliente (navegador) para desligar de esta actividad al servidor que recibirá los datos ingresados por el usuario.

Esta posibilidad de hacer pequeños programas que se ejecutan en el navegador, evitan intercambios innecesarios entre el cliente y el servidor (navegador y sitio web).
Suponemos que conoce las marcas para la creación de formularios en una página web:

form	<form> ... </form>
text	<input type="text">
password	<input type="password">
textarea	<textarea> ... </textarea>
button	<input type="button">
submit	<input type="submit">
reset	<input type="reset">
checkbox	<input type="checkbox">
radio	<input type="radio">
select	<select> ... </select>
hidden	<input type="hidden">

El navegador crea un objeto por cada control visual que aparece dentro de la página. Nosotros podemos acceder posteriormente desde JavaScript a dichos objetos. El objeto principal es el FORM que contendrá todos los otros objetos: TEXT (editor de líneas), TEXTAREA (editor de varias líneas), etc. Nuestra actividad en JavaScript es procesar los eventos que generan estos controles (un evento es una acción que se dispara, por ejemplo, si se presiona un botón).

Vamos a hacer un problema muy sencillo empleando el lenguaje Javascript; dispondremos un botón y cada vez que se presione, mostraremos un contador:

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
var contador=0;
function incrementar() {
  contador++;
  alert('El contador ahora vale :' + contador);
}
</script>

<form>
  <input type="button" onClick="incrementar()" value="incrementar">
</form>
</body>
</html>
```

A los eventos de los objetos HTML se les asocia una función, dicha función se ejecuta cuando se dispara el evento respectivo. En este caso cada vez que presionamos el botón, se llama a la función incrementar, en la misma incrementamos la variable contador en uno. Hay que tener en cuenta que a la variable contador la definimos fuera de la función para que no se inicialice cada vez que se dispara el evento.
La función alert crea una ventana que puede mostrar un mensaje.

Controles FORM, BUTTON y TEXT

Hasta ahora hemos visto como crear un formulario con controles de tipo BUTTON. Agregamos un control de tipo TEXT (permite al operador cargar caracteres por teclado).
Ahora veremos la importancia de definir un id a todo control de un formulario.
Con un ejemplo veremos estos controles: Confeccionar un formulario que permita ingresar el nombre y edad de una persona:

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

<script type="text/javascript">
function mostrar() {
  var nom=document.getElementById('nombre').value;
  var ed=document.getElementById('edad').value;
  alert('Ingresó el nombre:' + nom);
  alert('Y la edad:' + ed);
}
</script>

<form>
Ingrese su nombre:
<input type="text" id="nombre"><br>
Ingrese su edad:
<input type="text" id="edad"><br>
<input type="button" value="Confirmar" onClick="mostrar()">
</form>

</body>
</html>
```

En este problema tenemos cuatro controles: 1 FORM, 1 BUTTON, 2 TEXT.
El evento que se dispara al presionar el botón se llama mostrar.
La función 'mostrar' accede a los contenidos de los dos controles de tipo TEXT:

```
 var nom=document.getElementById('nombre').value;
```

```
 var ed=document.getElementById('edad').value;
```

Para hacer más clara la función guardamos en dos variables auxiliares los contenidos de los controles de tipo TEXT.
La propiedad **"id"** es un identificar único para cualquier marca HTML que luego nos permite desde Javascript acceder a dicho elemento.
El método **getElementById** nos retorna una referencia del objeto HTML que le pasamos como parámetro. A partir de este objeto accedemos a la propiedad value que almacena el valor ingresado por el operador en el control TEXT.
Luego de extraer los valores ingresados por el operador los mostramos utilizando la función alert:

```
 var nom=document.getElementById('nombre').value;
 var ed=document.getElementById('edad').value;
 alert('Ingresó el nombre:' + nom);
 alert('Y la edad:' + ed);
```

Control PASSWORD

Esta marca es una variante de la de tipo "TEXT". La diferencia fundamental es que cuando se carga un texto en el campo de edición sólo muestra asteriscos en pantalla, es decir, es fundamental para el ingreso de claves y para que otros usuarios no vean los caracteres que escribimos.

La mayoría de las veces este dato se procesa en el servidor. Pero podemos en el cliente (es decir en el navegador) verificar si ha ingresado una cantidad correcta de caracteres, por ejemplo.

Ejemplo: Codificar una página que permita ingresar una password y luego muestre una ventana de alerta si tiene menos de 5 caracteres.

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

<script type="text/javascript">
  function verificar(){
    var clave=document.getElementById('clave').value;
    if (clave.length<5) {
      alert('La clave no puede tener menos de 5 caracteres!!!');
    } else {
      alert('Largo de clave correcta');
    }
  }
```

```
</script>

<form>
Ingrese una clave:
<input type="password" id="clave">
<br>
<input type="button" value="Confirmar" onClick="verificar()">
</form>
</body>
</html>
```

En este problema debemos observar que cuando ingresamos caracteres dentro del campo de edición sólo vemos asteriscos, pero realmente en memoria se almacenan los caracteres tipeados. Si queremos mostrar los caracteres ingresados debemos acceder mediante el método getElementById a la marca HTML clave:

```
var clave=document.getElementById('clave').value;
```

Normalmente, a este valor no lo mostraremos dentro de la página, sino se perdería el objetivo de este control (ocultar los caracteres tipeados).
Si necesitamos saber la cantidad de caracteres que tiene un string accedemos a la propiedad length que retorna la cantidad de caracteres.

```
if (clave.length<5)
```

Control SELECT

Este otro objeto visual que podemos disponer en un FORM permite realizar la selección de un string de una lista y tener asociado al mismo un valor no visible. El objetivo fundamental en JavaScript es determinar qué elemento está seleccionado y qué valor tiene asociado. Esto lo hacemos cuando ocurre el evento *onChange*. Para determinar la posición del índice seleccionado en la lista:

```
document.getElementById('select1').selectedIndex;
```

Considerando que el objeto SELECT se llama select1 accedemos a la propiedad selectedIndex (almacena la posición del string seleccionado de la lista, numerando a partir de cero). Para determinar el string seleccionado:

```
document.getElementById('select1').options[document.getElementById('select1').selectedIndex].text;
```

Es decir que el objeto select1 tiene otra propiedad llamada options, a la que accedemos por medio de un subíndice, al string de una determinada

posición.

Hay problemas en los que solamente necesitaremos el string almacenado en el objeto SELECT y no el valor asociado (no es obligatorio asociar un valor a cada string).
Y por último con esta expresión accedemos al valor asociado al string:

```
document.getElementById('select1').options[document.getElementById('select1').selectedIndex].value;
```

Un ejemplo completo que muestra el empleo de un control SELECT es:

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

<script type="text/javascript">
function cambiarColor()
{
  var seleccion=document.getElementById('select1');
  document.getElementById('text1').value=seleccion.selectedIndex;
  document.getElementById('text2').value=seleccion.options[seleccion.selectedIndex].text;
  document.getElementById('text3').value=seleccion.options[seleccion.selectedIndex].value;
}
</script>

<form>
  <select id="select1" onChange="cambiarColor()">
    <option value="0xff0000">Rojo</option>
    <option value="0x00ff00">Verde</option>
    <option value="0x0000ff">Azul</option>
  </select>
  <br>
  Número de índice seleccionado del objeto SELECT:
  <input type="text" id="text1"><br>
  Texto seleccionado:<input type="text" id="text2"><br>
  Valor asociado:<input type="text" id="text3"><br>
</form>
</body>
</html>
```

Se debe analizar en profundidad este problema para comprender primeramente la creación del objeto SELECT en HTML, y cómo acceder luego a sus valores desde Javascript.
Es importante para el objeto SELECT definir qué función llamar cuando ocurra un cambio: onChange="cambiarColor()".
Por cada opción del objeto SELECT tenemos una línea:

Donde Rojo es el string que se visualiza en el objeto SELECT y value es el valor asociado a dicho string.

Analizando la función cambiarColor() podemos ver cómo obtenemos los valores fundamentales del objeto SELECT.

Control CHECKBOX

El control CHECKBOX es el cuadradito que puede tener dos estados (seleccionado o no seleccionado).
Para conocer su funcionamiento y ver cómo podemos acceder a su estado desde Javascript haremos una pequeña página.

Ejemplo: Confeccionar una página que muestre 4 lenguajes de programación que el usuario puede seleccionar si los conoce. Luego mostrar un mensaje indicando la cantidad de lenguajes que ha seleccionado el operador.

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

<script type="text/javascript">
function contarSeleccionados(){
  var cant=0;
  if (document.getElementById('checkbox1').checked) {
    cant++;
  }
  if (document.getElementById('checkbox2').checked) {
    cant++;
  }
  if (document.getElementById('checkbox3').checked) {
    cant++;
  }
  if (document.getElementById('checkbox4').checked) {
    cant++;
  }
  alert('Conoce ' + cant + ' lenguajes');
}
</script>

<form>
<input type="checkbox" id="checkbox1">JavaScript
<br>
<input type="checkbox" id="checkbox2">PHP
<br>
<input type="checkbox" id="checkbox3">JSP
<br>
<input type="checkbox" id="checkbox4">VB.Net
```

```
<br>
<input type="button" value="Mostrar" onClick="contarSeleccionados()">
</form>
</body>
</html>
```

Cuando se presiona el botón se llama a la función Javascript contarSeleccionados(). En la misma verificamos uno a uno cada control checkbox accediendo a la propiedad checked del elemento que almacena true o false según esté o no seleccionado el control:
Disponemos un 'if' para cada checkbox:

```
if (document.getElementById('checkbox1').checked)  {
  cant++;
}
```

Como la propiedad checked almacena un true o false podemos utilizar dicho valor directamente como valor de la condición en lugar de codificar:

```
if (document.getElementById('checkbox1').checked==true)  {
  cant++;
}
```

Al contador 'cant' lo definimos e inicializamos en cero previo a los cuatro if.
Mostramos finalmente el resultado final.

Control RADIO

Los objetos RADIO tienen sentido cuando disponemos varios elementos y solo uno puede estar seleccionado del conjunto.
Ejemplo: Mostrar cuatro objetos de tipo RADIO que permitan seleccionar los estudios que tiene un usuario:

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

```
<body>
<script type="text/javascript">
  function mostrarSeleccionado(){
    if (document.getElementById('radio1').checked){
      alert('no tienes estudios');
    }
    if (document.getElementById('radio2').checked){
      alert('tienes estudios primarios');
    }
    if (document.getElementById('radio3').checked){
      alert('tienes estudios secundarios');
    }
    if (document.getElementById('radio4').checked){
      alert('tienes estudios universitarios');
    }
  }
</script>
<form>
<input type="radio" id="radio1" name="estudios">Sin estudios
<br>
<input type="radio" id="radio2" name="estudios">Primarios
<br>
<input type="radio" id="radio3" name="estudios">Secundarios
<br>
<input type="radio" id="radio4" name="estudios">Universitarios
<br>
<input type="button" value="Mostrar" onClick="mostrarSeleccionado()">
</form>
</body>
</html>
```

Es importante notar que todos los objetos de tipo RADIO tienen definida la propiedad name con el mismo valor (esto permite especificar que queremos que los radios estén relacionados entre sí).
Luego podemos acceder a cada elemento mediante el método getElementById para consultar la propiedad checked:

```
  if (document.getElementById('radio1').checked)   {
     alert('no tienes estudios');
   }
```

Igual que el checkbox, la propiedad checked retorna true o false, según esté o no seleccionado el control radio.

Cuando se dispone de un grupo de *radiobuttons*, generalmente no se quiere obtener el valor del atributo value de alguno de ellos, sino que lo importante es conocer cuál de todos los *radiobuttons* se ha seleccionado. La propiedad checked devuelve true para el *radiobutton* seleccionado y false en cualquier otro caso. Si por ejemplo se dispone del siguiente grupo de *radiobuttons*:

```
<input type="radio" value="si" name="pregunta" id="pregunta_si"/> SI
<input type="radio" value="no" name="pregunta" id="pregunta_no"/> NO
<input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc"/> NS/NC
```

El siguiente código permite determinar si cada *radiobutton* ha sido seleccionado o no:

```
var elementos = document.getElementsByName("pregunta");

for(var i=0; i<elementos.length; i++) {
  alert(" Elemento: " + elementos[i].value + "\n Seleccionado: " + elementos[i].checked);
}
```

Control TEXTAREA

Este control es similar al control TEXT, salvo que permite el ingreso de muchas líneas de texto. La marca TEXTAREA en HTML tiene dos propiedades: rows y cols que nos permiten indicar la cantidad de filas y columnas a mostrar en pantalla.
Ejemplo: Solicitar la carga del mail y el curriculum de una persona. Mostrar un mensaje si el curriculum supera los 2000 caracteres.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
function controlarCaracteres(){
  if (document.getElementById('curriculum').value.length>2000)  {
    alert('curriculum muy largo');
  } else {
    alert('datos correctos');
  }
}
</script>
<form>
<textarea id="curriculum" rows="10" cols="50" ></textarea>
<br>
<input type="button" value="Mostrar" onClick="controlarCaracteres()">
</form>
</body>
</html>
```

Para saber el largo de la cadena cargada:

```
  if (document.getElementById('curriculum').value.length>2000)
```

accedemos a la propiedad length.

El objeto window

Al objeto window lo hemos estado usando constantemente. Representa la ventana del navegador.
window es un objeto global y tiene los siguienes métodos:

alert: Muestra un diálogo de alerta con un mensaje (la hemos utilizado desde los primeros temas)
prompt: Muestra un diálogo para la entrada de un valor de tipo string (utilizado desde el primer momento)
confirm: Muestra un diálogo de confirmación con los botones Confirmar y Cancelar.
open y close: abre o cierra una ventana del navegador. Podemos especificar el tamaño de la ventana, su contenido, etc.

```
[Variable=][window.]open(URL, nombre, propiedades)
```

Permite crear (y abrir) una nueva ventana. Si queremos tener acceso a ella desde la ventana donde la creamos, deberemos asignarle una variable, sino simplemente invocamos el método: el navegador automáticamente sabrá que pertenece al objeto window.
El parámetro URL es una cadena que contendrá la dirección de la ventana que estamos abriendo: si está en blanco, la ventana se abrirá con una página en blanco.
Las propiedades son una lista, separada por comas, de algunos de los siguientes elementos:

```
toolbar[=yes|no]
location[=yes|no]
directories[=yes|no]
status[=yes|no]
menubar[=yes|no]
scrollbars[=yes|no]
resizable[=yes|no]
width=pixels
height=pixels
```

Es bueno hacer notar que a todas estas funciones las podemos llamar anteponiéndole el nombre del objeto window, seguida del método o en forma resumida indicando solamente el nombre del método (como lo hemos estado haciendo), esto es posible ya que el objeto window es el objeto de

máximo nivel.

```
valor=window.prompt("Ingrese valor","");
```

o

```
valor=prompt("Ingrese valor","");
```

Para reducir la cantidad de caracteres que se tipean normalmente encontraremos los programas tipeados de la segunda forma.

El siguiente programa muestra varios de los métodos disponibles del objeto window:

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

<script type="text/javascript">

 function abrir()
 {
   var ventana=open();
   ventana.document.write("Estoy escribiendo en la nueva ventana<br>");
   ventana.document.write("Segunda linea");
 }

 function abrirParametros()
 {
   //open(URL, nombre, propiedades)
   var ventana=open('','','status=yes,width=400,height=250,menubar=yes');
   ventana.document.write("Esto es lo primero que aparece<br>");
 }

 function mostrarAlerta()
 {
   alert("Esta ventana de alerta ya la utilizamos en otros problemas.");
 }

 function confirmar()
 {
   var respuesta=confirm("Presione alguno de los dos botones");
   if (respuesta==true)
     alert("presionó aceptar");
   else
     alert("presionó cancelar");
 }

 function cargarCadena()
 {
   var cad=prompt("cargue una cadena:","");
   alert("Usted ingreso "+cad);
 }
</script>
```

```
Este programa permite analizar la llamada a distintas responsabilidades del objeto window.<br>
<form>
  <br>
  <input type="button" value="open()" onClick="abrir()">
  <br>
  <input type="button" value="open con parámetros" onClick="abrirParametros()" >
  <br>
  <input type="button" value="alert" onClick="mostrarAlerta()">
  <br>
  <input type="button" value="confirm" onClick="confirmar()">
  <br>
  <input type="button" value="prompt" onClick="cargarCadena()">
</form>
</body>
</html>
```

Archivo JavaScript externo (*.js)

El lenguaje JavaScript permite agrupar funciones y disponerlas en un archivo separado a la página HTML. Esto trae muchos beneficios:
- Reutilización de funciones en muchos archivos. No tenemos que copiar y pegar sucesivamente las funciones en las páginas en las que necesitamos.
- Facilita el mantenimiento de las funciones al encontrarse en archivos separados.
- Nos obliga a ser más ordenados.
La mecánica para implementar estos archivos externos en JavaScript es:

1 - Crear un archivo con extensión *.js y escribir las funciones en la misma:

```
function retornarFecha(){
  var fecha
  fecha=new Date();
  var cadena=fecha.getDate()+'/'+(fecha.getMonth()+1)+'/'+fecha.getYear();
  return cadena;
}

function retornarHora(){
  var fecha
  fecha=new Date();
  var cadena=fecha.getHours()+':'+fecha.getMinutes()+':'+fecha.getSeconds();
  return cadena;
}
```

2 - Creamos un archivo html que utilizará las funciones contenidas en el archivo *.js:

```
<html>
<head>
<title>Problema</title>
<script type="text/javascript" src="funciones.js"></script>
</head>
<body>
<script type="text/javascript">
  document.write('La fecha de hoy es:'+retornarFecha());
  document.write('<br>');
  document.write('La hora es:'+retornarHora());
</script>
</body>
</html>
```

Es decir debemos disponer el siguiente código para importar el archivo *.js:

```
<script type="text/javascript" src="funciones.js"></script>
```

Mediante la propiedad src indicamos el nombre del archivo a importar.
Luego, podemos llamar dentro de la página HTML, a las funciones que contiene el archivo externo *.js; en nuestro ejemplo llamamos a las funciones retornarFecha() y retornarHora().
Como podemos ver, el archivo html queda mucho más limpio.

Eventos

Como hemos visto hasta ahora uno de los usos principales de JavaScript es la implementación de algoritmos que reaccionan a eventos que se producen en una página web. Cuando se termina de cargar completamente una página web se dispara el evento onLoad, cuando se presiona un botón de tipo submit se dispara el evento onsubmit, cuando movemos la flecha del mouse se dispara onMouseMove y otra gran cantidad de eventos que nos informa el navegador y nosotros podemos capturarlos y hacer un algoritmo a nuestra medida. JavaScript implementa tres formas distintas de capturar los eventos que emite el navegador.

1. Eventos definidos directamente en la marca HTML.
Esta metodología está en desuso, pero muchos sitios creados a fines de 1990 y principios del 2000 implementan esta técnica.

Problema: Implementar un formulario que solicite la carga del nombre de usuario y su clave. Mostrar un mensaje si no se ingresan datos en alguno de los dos controles.

```
<html>
<head>
<script type="text/javascript">
  function validar()
  {
     var usu=document.getElementById("usuario").value;
     var cla=document.getElementById("clave").value;
     if (usu.length==0 || cla.length==0)
     {
       alert('El nombre de usuario o clave está vacío');
       return false;
     }
     else
       return true;
  }
</script>
</head>
<body>

<form method="post" action="procesar.php" onsubmit="validar();" id="formulario1">
Ingrese nombre:
<input type="text" id="usuario" name="usuario" size="20">
<br>
Ingrese clave:
<input type="password" id="clave" name="clave" size="20">
<br>
<input type="submit" id="confirmar" name="confirmar" value="Confirmar">
</form>
</body>
</html>
```

Lo primero que tenemos que ver que la marca form define la propiedad onsubmit y le asigna el nombre de la función JavaScript que debe llamarse previo a que el navegador empaquete todos los datos del formulario y los envía al servidor para ser procesados:

```
<form method="post" action="procesar.php" onsubmit="validar();">
```

Como vemos debemos indicar el nombre de la función y los paréntesis (en este caso no se envían parámetros por lo que los paréntesis van abiertos y cerrados). En el bloque JavaScript debemos implementar la función validar donde extraemos los valores de cada control y verificamos si alguno de los dos no tiene cargado caracteres, en caso que suceda esto mostramos un mensaje y retornamos un false para que el navegador no envíe los datos del formulario al servidor. Si la función retorna true los datos del formulario son enviados por el navegador al servidor:

```
<script type="text/javascript">

  function validar()  {
```

```
   var usu=document.getElementById("usuario").value;
   var cla=document.getElementById("clave").value;
   if (usu.length==0 || cla.length==0)  {
     alert('El nombre de usuario o clave está vacío');
     return false;
   }  else
     return true;
 }

</script>
```

Esta metodología de inicializar eventos debe ser evitada en todo lo posible.

2. Eventos definidos accediendo a propiedades del objeto.
Esta metodología es todavía ampliamente utilizada ya que la mayoría de los navegadores lo implementan en forma similar.

Problema: Implementar un formulario que solicite la carga del nombre de usuario y su clave. Mostrar un mensaje si no se ingresan datos en alguno de los dos controles.

```
<html>
<head>

<script type="text/javascript">

 window.onload=inicio;

 function inicio()
 {
   document.getElementById("formulario1").onsubmit=validar;
 }

 function validar()
 {
    var usu=document.getElementById("usuario").value;
    var cla=document.getElementById("clave").value;
    if (usu.length==0 || cla.length==0)
    {
      alert('El nombre de usuario o clave está vacío');
      return false;
    }
    else
      return true;
 }

</script>

</head>
<body>

<form method="post" action="procesar.php" id="formulario1">
Ingrese nombre:
```

```
<input type="text" id="usuario" name="usuario" size="20">
<br>
Ingrese clave:
<input type="password" id="clave" name="clave" size="20">
<br>
<input type="submit" id="confirmar" name="confirmar" value="Confirmar">
</form>

</body>
</html>
```

Con esta segunda metodología vemos que el código HTML queda limpio de llamadas a funciones JavaScript y todo el código queda dentro del bloque del script (pudiendo luego llevar todo este bloque a un archivo externo *.js) Lo primero que vemos es inicializar la propiedad onload del objeto window con el nombre de la función que se ejecutará cuando finalice la carga completa de la página, es importante notar que a la propiedad onload le asignamos el nombre de la función y NO debemos disponer los paréntesis abiertos y cerrados (ya que no se está llamando a la función sino le estamos pasando la dirección o referencia de la misma)

```
window.onload=inicio;
```

La función inicio es llamada por el objeto window cuando se termina de cargar la página. En esta función obtenemos la referencia del objeto formulario1 mediante el método getElementById e inicializamos la propiedad onsubmit con el nombre de la función que será llamada cuando se presione el botón submit del formulario:

```
function inicio()   {
  document.getElementById("formulario1").onsubmit=validar;
}
```

Por último, tenemos la función validar que verifica si los dos controles del formulario están cargados:

```
function validar()   {
   var usu=document.getElementById("usuario").value;
   var cla=document.getElementById("clave").value;
   if (usu.length==0 || cla.length==0)
   {
     alert('El nombre de usuario o clave está vacío');
     return false;
   }
   else
     return true;
}
```

La misma metodología, pero utilizando funciones anónimas para cada evento el código ahora queda condensado:

```
<html>
<head>
<script type="text/javascript">

  window.onload=function() {
    document.getElementById("formulario1").onsubmit=function () {
      var usu=document.getElementById("usuario").value;
      var cla=document.getElementById("clave").value;
      if (usu.length==0 || cla.length==0)      {
        alert('El nombre de usuario o clave está vacío');
        return false;
      } else
        return true;
    }
  }
</script>

</head>
<body>

<form method="post" action="procesar.php" id="formulario1">
Ingrese nombre:
<input type="text" id="usuario" name="usuario" size="20">
<br>
Ingrese clave:
<input type="password" id="clave" name="clave" size="20">
<br>
<input type="submit" id="confirmar" name="confirmar" value="Confirmar">
</form>

</body>
</html>
```

Analicemos un poco el código implementado, a la propiedad onload del objeto window le asignamos una función anónima:

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

En la implementación de la función anónima inicializamos la propiedad onsubmit del objeto formulario1 con otra función anónima:

```
document.getElementById("formulario1").onsubmit=function () {
     ...
   }
```

Esta sintaxis de funciones anónimas es ampliamente utilizado.

Modelo de eventos definidos por W3C (World Wide Web Consortium)
Este modelo de eventos se basa en la implementación de un método para todos los objetos que nos permite registrar eventos. La sintaxis del método es:

```
addEventListener(evento, método a ejecutar, fase);
```

Veamos como implementamos el problema anterior utilizando este nuevo modelo de eventos:

```
<html>
<head>

<script type="text/javascript">

 window.addEventListener('load',inicio,false);

 function inicio(){
document.getElementById("formulario1").addEventListener('submit',validar,false);
 }

 function validar(evt){
    var usu=document.getElementById("usuario").value;
    var cla=document.getElementById("clave").value;
    if (usu.length==0 || cla.length==0){
      alert('El nombre de usuario o clave está vacío');
      evt.preventDefault();
    }
 }
</script>

</head>
<body>
<form method="post" action="procesar.php" id="formulario1">
Ingrese nombre:
<input type="text" id="usuario" name="usuario" size="20">
<br>
Ingrese clave:
<input type="password" id="clave" name="clave" size="20">
<br>
<input type="submit" id="confirmar" name="confirmar" value="Confirmar">
</form>
</body>
</html>
```

Lo primero que vemos es que en vez de inicializar la propiedad onload procedemos a llamar al método addEventListener:

```
window.addEventListener('load',inicio,false);
```

El primer parámetro es un string con el nombre del evento a inicializar, el segundo parámetro es el nombre de la función a ejecutar y el tercer parámetro normalmente es el valor false.
Cuando se carga completamente la página el objeto window tiene la referencia al método que se debe llamar, en nuestro caso se llama inicio. La función inicio obtiene la referencia del objeto formulario1 y procede a registrar el evento submit indicando en el segundo parámetro el nombre de la función que debe ejecutarse:

```
function inicio()  {
      document.getElementById("formulario1").addEventListener('submit',validar,false);
   }
```

El código de la función validar se modifica, llega como parámetro una referencia al evento y mediante este llamamos al método preventDefault si queremos que no se envíen los datos al servidor:

```
function validar(evt)  {
   var usu=document.getElementById("usuario").value;
   var cla=document.getElementById("clave").value;
   if (usu.length==0 || cla.length==0)
   {
     alert('El nombre de usuario o clave está vacío');
     evt.preventDefault();
   }
}
```

Este modelo de eventos está siendo ampliamente implementado por los navegadores modernos. El problema es el IE8 o inferiores que no lo implementa de esta forma.

Eventos onFocus y onBlur

El evento onFocus se dispara cuando el objeto toma foco y el evento onBlur cuando el objeto pierde el foco.

Ejemplo: Implementar un formulario que solicite la carga del nombre y la edad de una persona. Cuando el control tome foco borrar el contenido actual, al abandonar el mismo, mostrar un mensaje de alerta si el mismo está vacío. Mostrar en las propiedades value de los controles text los mensajes "nombre" y "mail" respectivamente.

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

```
<script type="text/javascript">
function vaciar(control){
  control.value='';
}
function verificarEntrada(control){
  if (control.value=='')
    alert('Debe ingresar datos');
}
</script>
<form>
<input type="text" id="nombre" onFocus="vaciar(this)" onBlur="verificarEntrada(this)"
value="nombre"><br>
<input type="text" id="edad" onFocus="vaciar(this)" onBlur="verificarEntrada(this)" value="mail">
<br>
<input type="button" value="Confirmar">
</form>
</body>
</html>
```

A cada control de tipo TEXT le inicializamos los eventos onFocus y onBlur. También cargamos las propiedades *value* para mostrar un texto dentro del control. Le indicamos, para el evento onFocus la función vaciar, pasando como parámetro la palabra clave this que significa la dirección del objeto que emitió el evento. En la función propiamente dicha, accedemos a la propiedad value y borramos su contenido. Esto nos permite definir una única función para vaciar los dos controles.
De forma similar, para el evento onBlur llamamos a la función verificarEntrada donde analizamos si se ha ingresado algún valor dentro del control, en caso de tener un string vacío procedemos a mostrar una ventana de alerta.

Evento onLoad

El evento onLoad se ejecuta cuando cargamos una página en el navegador. Uno de los usos más frecuentes es para fijar el foco en algún control de un formulario, para que el operador no tenga que activar con el mouse dicho control. Este evento está asociado a la marca body.

La página completa es:

```
<html>
<head></head>
<body onLoad="activarPrimerControl()">

<script type="text/javascript">
  function activarPrimerControl()
  {
    document.getElementById('nombre').focus();
```

```
  }
</script>

<form>
  Ingrese su nombre:
  <input type="text" id="nombre"><br>
  Ingrese su edad:
  <input type="text" id="edad"><br>
  <input type="button" value="Confirmar">
</form>
</body>
</html>
```

En la marca body inicializamos el evento onLoad con la llamada a la función activarPrimerControl:

```
<body onLoad="activarPrimerControl()">
```

La función da el foco al control text donde se cargará el nombre:

```
 function activarPrimerControl()  {
   document.getElementById('nombre').focus();
 }
```

Eventos: click y dblclick

Dos eventos que podemos capturar de cualquier elemento HTML son cuando un usuario hace un clic o un doble clic sobre el mismo.
El evento click se produce cuando el usuario hace un solo clic sobre el elemento HTML y suelta inmediatamente el botón del mouse en el mismo lugar y el dblclick cuando presiona en forma sucesiva en la misma ubicación.

Para probar como registramos estos eventos implementaremos una página que muestre dos div y definiremos el evento click para el primero y el evento dblclick para el segundo.

```
<html>
<head>

<script type="text/javascript">

  addEventListener('load',inicio,false);

  function inicio() {
    document.getElementById('recuadro1').addEventListener('click',presion1,false);
    document.getElementById('recuadro2').addEventListener('dblclick',presion2,false);
```

```
  }

  function presion1() {
    alert('se hizo click');
  }

  function presion2() {
    alert('se hizo doble click');
  }

</script>
</head>
<body>

<div style="width:200px;height:200px;background:#ffff00" id="recuadro1">
   Prueba del evento click
</div>
<div style="width:200px;height:200px;background:#ff5500" id="recuadro2">
   Prueba del evento dblclick
</div>

</body>
</html>
```

Primero registramos mediante la llamada al método addEventListener el evento load de la página (recordemos que esta forma de registrar los eventos no funciona en el IE8 o anteriores, pero en muy poco tiempo la cantidad de dispositivos con dichos navegadores serán ínfimas), y dentro de la función inicio registramos los eventos click y dblclick para los dos div que definimos en la página HTML:

```
addEventListener('load',inicio,false);

function inicio()   {
   document.getElementById('recuadro1').addEventListener('click',presion1,false);
   document.getElementById('recuadro2').addEventListener('dblclick',presion2,false);
}
```

Como vemos lo único que hacemos en los métodos que se disparan es mostrar un mensaje:

```
  function presion1()  {
        alert('se hizo click');
     }

     function presion2()  {
        alert('se hizo doble click');
     }
```

Eventos: mousedown y mouseup

Otros dos eventos relacionados con el mouse son mousedown y mouseup. El evento mousedown se dispara inmediatamente luego que presionamos con la flecha del mouse un elemento HTML que tiene registrado dicho evento. El evento mouseup se ejecuta luego de soltar el botón del mouse estando dentro del control HTML.

Para probar estos eventos implementaremos una página que contenga dos div, a un div le asignaremos el evento mousedown y al otro el evento mouseup. Cuando ocurra el evento procederemos a cambiar el texto contenido dentro del div.

```
<html>
<head>

<script type="text/javascript">

  addEventListener('load',inicio,false);

  function inicio()
  {
    document.getElementById('recuadro1').addEventListener('mousedown',presion1,false);
    document.getElementById('recuadro2').addEventListener('mouseup',presion2,false);
  }

  function presion1()
  {
    document.getElementById('recuadro1').innerHTML='Se presione el mouse y todavía no se
soltó';
  }

  function presion2()
  {
    document.getElementById('recuadro2').innerHTML='Se presione el mouse y se soltó';
  }

</script>

</head>
<body>
<p>Presione el recuadro amarillo sin soltar el botón del mouse.</p>
<div style="width:200px;height:200px;background:#ffff00" id="recuadro1">
</div>
<p>Presione el recuadro naranja y suelte el botón del mouse.</p>
<div style="width:200px;height:200px;background:#ff5500" id="recuadro2">
</div>

</body>
</html>
```

En la función inicio registramos los eventos mousedown y mouseup para los dos div, al div recuadro1 procedemos a registrar el evento mousedown y al div recuadro2 procedemos a registrar el evento mouseup:

```
function inicio()  {
  document.getElementById('recuadro1').addEventListener('mousedown',presion1,false);
  document.getElementById('recuadro2').addEventListener('mouseup',presion2,false);
}
```

El método que se dispara para el primer div procedemos a modificar todo el contenido del div accediendo a la propiedad innerHTML:

```
function presion1()  {
  document.getElementById('recuadro1').innerHTML='Se presione el mouse y todavía no se soltó';
}
```

De forma similar codificamos la función que modifica el segundo div:

```
function presion2()  {
  document.getElementById('recuadro2').innerHTML='Se presione el mouse y se soltó';
}
```

Eventos onMouseOver y onMouseOut

El evento onMouseOver se ejecuta cuando pasamos la flecha del mouse sobre un elemento HTML y el evento onMouseOut cuando la flecha abandona el mismo.

Para probar estos eventos implementaremos una página que cambie el color de fondo del documento.

Implementaremos una función que cambie el color con un valor que llegue como parámetro. Cuando retiramos la flecha del mouse volvemos a pintar de blanco el fondo del documento:

```
<html>
<head></head>
<body>
<script type="text/javascript">
  function pintar(col) {
    document.bgColor=col;
  }
</script>
<a href="pagina1.html" onMouseOver="pintar('#ff0000')" onMouseOut="pintar('#ffffff')">Rojo</a>
```

```
-
<a href="pagina1.html" onMouseOver="pintar('#00ff00')" onMouseOut="pintar('#ffffff')">Verde</a>
-
<a href="pagina1.html" onMouseOver="pintar('#0000ff')" onMouseOut="pintar('#ffffff')">Azul</a>
-
<input type="text" onMouseOver="pintar('#000000')" onMouseOut="pintar('#ffffff')" value="negro">
</body>
</html>
```

Las llamadas a las funciones las hacemos inicializando las propiedades onMouseOver y onMouseOut:

```
<a href="pagina1.html" onMouseOver="pintar('#ff0000')" onMouseOut="pintar('#ffffff')">Rojo</a>
```

La función 'pintar' recibe el color e inicializa la propiedad bgColor del objeto document.

```
function pintar(col)  {
  document.bgColor=col;
}
```

Otro problema que podemos probar es pintar de color el interior de una casilla de una tabla y regresar a su color original cuando salimos de la misma:

```
<html>
<head></head>
<body>
<script type="text/javascript">
  function pintar(objeto,col)   {
    objeto.bgColor=col;
  }
</script>
<table border="1">
<tr>
<td onMouseOver="pintar(this,'#ff0000')" onMouseOut="pintar(this,'#ffffff')">rojo</td>
<td onMouseOver="pintar(this,'#00ff00')" onMouseOut="pintar(this,'#ffffff')">verde</td>
<td onMouseOver="pintar(this,'#0000ff')" onMouseOut="pintar(this,'#ffffff')">azul</td>
</tr>
</table>
</body>
</html>
```

La lógica es bastante parecida a la del primer problema, pero en éste, le pasamos como parámetro a la función, la referencia a la casilla que queremos que se coloree (**this**):

```
<td onMouseOver="pintar(this,'#ff0000')" onMouseOut="pintar(this,'#ffffff')">rojo</td>
```

Eventos: mouseover y mouseout

El evento mouseover se dispara cuando ingresamos con la flecha del mouse a un control HTML que tiene registrado dicho evento, y mouseout se dispara cuando sacamos la flecha del mouse del control.

Es muy común utilizar estos dos eventos para producir cambios en el elemento HTML cuando ingresamos la flecha del mouse y retornar al estado anterior cuando se saca la flecha del mouse.

Es importante hacer notar que estos eventos se disparan sin tener que presionar la flecha del mouse, solo con desplazarla al interior del elemento HTML se dispara el evento mouseover.

Implementaremos un pequeño ejemplo que muestre un cuadrado, el mismo mostrará los bordes redondeados cuando ingresemos la flecha del mouse en su interior y volverá al estado anterior cuando retiremos la flecha.

```
<html>
<head>

<script type="text/javascript">

 addEventListener('load',inicio,false);

 function inicio()
 {
  document.getElementById('recuadro1').addEventListener('mouseover',entrada,false);
  document.getElementById('recuadro1').addEventListener('mouseout',salida,false);
 }

 function entrada()
 {
  document.getElementById('recuadro1').style.borderRadius='30px';
 }

 function salida()
 {
  document.getElementById('recuadro1').style.borderRadius='0px';
 }

</script>

</head>
<body>
<div style="width:200px;height:200px;background:#0000ff" id="recuadro1">
</div>

</body>
</html>
```

Definimos en el HTML un div de color azul de 200 píxeles de lado:

```
<div style="width:200px;height:200px;background:#0000ff" id="recuadro1">
</div>
```

En la función inicio registramos los eventos mouseover y mouseout:

```
function inicio()   {
   document.getElementById('recuadro1').addEventListener('mouseover',entrada,false);
   document.getElementById('recuadro1').addEventListener('mouseout',salida,false);
 }
```

El método entrada se ejecuta cuando ingresemos la flecha del mouse en el div llamado recuadro1 y procedemos en el mismo a modificar la propiedad borderRadius del estilo de este elemento (indicamos que el redondeo de los vértices del div sea de 30 píxeles):

```
 function entrada()   {
     document.getElementById('recuadro1').style.borderRadius='30px';
   }
```

Cuando sale la flecha del mouse del div se ejecuta la función salida donde fijamos con 0 la propiedad borderRadius:

```
 function salida() {
   document.getElementById('recuadro1').style.borderRadius='0px';
 }
```

Evento: mousemove

El evento mousemove se dispara cada vez que desplazamos la flecha del mouse sobre el elemento HTML que esta escuchando este evento.

Este tipo de evento hay que utilizarlo con cuidado ya que puede sobrecargar el navegador, ya que este evento se dispara cada vez que desplazamos aunque sea solo un pixel.

Implementaremos para ver su funcionamiento una página que muestre un div que capture el evento mousemove y como acción incrementaremos un contador para saber la cantidad de veces que se disparó el evento.

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

```
<script type="text/javascript">

  addEventListener('load',inicio,false);

  function inicio(){
    document.getElementById('recuadro1').addEventListener('mousemove',mover,false);
  }

  function mover()
  {
    var x=parseInt(document.getElementById('cantidad').innerHTML);
    x++;
    document.getElementById('cantidad').innerHTML=x;
  }

</script>

</head>
<body>
<div style="width:200px;height:200px;background:#0000ff" id="recuadro1">
</div>
<p id="cantidad">0</p>

</body>
</html>
```

Disponemos un div y un párrafo donde mostramos el número 0:

```
<div style="width:200px;height:200px;background:#0000ff" id="recuadro1">
</div>
<p id="cantidad">0</p>
```

Capturamos el evento mousemove para el div:

```
  function inicio()  {
      document.getElementById('recuadro1').addEventListener('mousemove',mover,false);
    }
```

Cada vez que se emite el evento mousemove se llama el método mover donde extraemos el valor del párrafo, lo incrementamos en uno y lo volvemos a actualizar:

```
  function mover()  {
        var x=parseInt(document.getElementById('cantidad').innerHTML);
        x++;
        document.getElementById('cantidad').innerHTML=x;
      }
```

Eventos: keydown, keyup y keypress

Estos tres eventos son similares a los eventos del mouse: mousedown, mouseup y click pero orientados al teclado del equipo.

El evento keydown se dispara cuando presionamos cualquier tecla del teclado, el evento keyup cuando soltamos una tecla. En cuanto el evento keypress en un principio procesa tanto cuando se la presionó y soltó, el único y gran inconveniente es que la mayoría de los navegadores no dispara el evento keypress para todas las teclas del teclado.

Para probar el evento keyup implementaremos un programa que permita solo ingresar 140 caracteres y nos informe con un mensaje la cantidad de caracteres disponibles para seguir escribiendo.

```
<html>
<head>

<script type="text/javascript">

  addEventListener('load',inicio,false);

  function inicio(){
    document.getElementById('texto').addEventListener('keyup',presion,false);
  }

  function presion(){
    var canti=document.getElementById('texto').value.length;
    var disponibles=140-parseInt(canti);
    document.getElementById('cantidad').innerHTML=disponibles;
  }

</script>

</head>
<body>
<input type="text" id="texto" maxlength="140" size="140">
<br>
<p>Máxama cantidad de caracteres disponibles:<span id="cantidad">140</span></p>
</body>
</html>
```

Definimos un control text y lo limitamos a 140 como máximo:

```
<input type="text" id="texto" maxlength="140" size="140">
```

Disponemos un elemento span para mostrar la cantidad de caracteres como máximo a ingresar:

```
<p>Máxama cantidad de caracteres disponibles:<span id="cantidad">140</span></p>
```

Registramos el evento keyup para el control texto:

```
function inicio()  {
  document.getElementById('texto').addEventListener('keyup',presion,false);
}
```

Cada vez que se suelta la tecla cuando estamos escribiendo en el control de texto procedemos a extraer el valor del control texto y obtener mediante la propiedad length la cantidad de caracteres tipeados hasta este momento y seguidamente restamos a 140 el número de caractes tipeados y procedemos a mostrarlo en el elemento spam:

```
function presion()   {
  var canti=document.getElementById('texto').value.length;
  var disponibles=140-parseInt(canti);
  document.getElementById('cantidad').innerHTML=disponibles;
}
```

Evento: change

El evento change se lo puede asociar a distintos elementos de formularios HTML y su comportamiento depende de cada uno.

Cuando asociamos el evento change a un control de tipo checkbox el mismo se dispara inmediatamente después que lo chequeamos o sacamos la selección. Para los controles de tipo radio también se dispara luego que cambia el estado de selección del mismo.

Para los controles select se dispara el evento change cuando cambiamos el item seleccionado.

Para los controles text y textarea se dispara cuando pierde el foco del control (porque seleccionamos otro control) y hemos cambiando el contenido del mismo.

Problema: Implementaremos una aplicación que muestre un alert cada vez que sucede el evento change para los controles checkbox, radio, select, text y textarea.

```
<html>
```

```
<head>

<script type="text/javascript">

 addEventListener('load',inicio,false);

 function inicio(){
   document.getElementById('checkbox1').addEventListener('change',cambiocheckbox,false);
   document.getElementById('radioa').addEventListener('change',cambioradio,false);
   document.getElementById('radiob').addEventListener('change',cambioradio,false);
   document.getElementById('radioc').addEventListener('change',cambioradio,false);
   document.getElementById('select1').addEventListener('change',cambioselect,false);
   document.getElementById('text1').addEventListener('change',cambiotext,false);
   document.getElementById('textarea1').addEventListener('change',cambiotextarea,false);
 }

 function cambiocheckbox(){
  alert(document.getElementById('checkbox1').checked);
 }

 function cambioradio(){
   alert(document.getElementById('radioa').checked+' '+
         document.getElementById('radiob').checked+' '+
         document.getElementById('radioc').checked);
 }

 function cambioselect(){
   alert(document.getElementById('select1').value);
 }

 function cambiotext(){
   alert(document.getElementById('text1').value);
 }

 function cambiotextarea(){
   alert(document.getElementById('textarea1').value);
 }

</script>

</head>
<body>
Evento change del checkbox:
<input type="checkbox" id="checkbox1" name="checkbox1" checked>
<br>
Evento change del radio:<br>
<input type="radio" name="radio1" id="radioa"><br>
<input type="radio" name="radio1" id="radiob"><br>
<input type="radio" name="radio1" id="radioc"><br>
Evento change del select:
<select name="select1" id="select1">
 <option value="argentina">Argentina</option>
 <option value="chile">Chile</option>
 <option value="uruguay">Uruguay</option>
 <option value="paraguay">Paraguay</option>
 <option value="bolivia">Bolivia</option>
```

```
</select>
<br>
<input type="text" name="text1" id="text1">
<br>
<textarea name="text1" id="textarea1" rows="6" cols="80"></textarea>
<br>
</body>
</html>
```

Creamos una serie de controles de formulario HTML en el body de la página:

```
Evento change del checkbox:
<input type="checkbox" id="checkbox1" name="checkbox1" checked>
<br>
Evento change del radio:<br>
<input type="radio" name="radio1" id="radioa"><br>
<input type="radio" name="radio1" id="radiob"><br>
<input type="radio" name="radio1" id="radioc"><br>
Evento change del select:
<select name="select1" id="select1">
  <option value="argentina">Argentina</option>
  <option value="chile">Chile</option>
  <option value="uruguay">Uruguay</option>
  <option value="paraguay">Paraguay</option>
  <option value="bolivia">Bolivia</option>
</select>
<br>
<input type="text" name="text1" id="text1">
<br>
<textarea name="text1" id="textarea1" rows="6" cols="80"></textarea>
<br>
```

En la función inicio asociamos a todos los controles para el evento change un método:

```
  function inicio() {
        document.getElementById('checkbox1').addEventListener('change',cambiocheckbox,false);
        document.getElementById('radioa').addEventListener('change',cambioradio,false);
        document.getElementById('radiob').addEventListener('change',cambioradio,false);
        document.getElementById('radioc').addEventListener('change',cambioradio,false);
        document.getElementById('select1').addEventListener('change',cambioselect,false);
        document.getElementById('text1').addEventListener('change',cambiotext,false);
        document.getElementById('textarea1').addEventListener('change',cambiotextarea,false);
    }
```

La función cambiocheckbox se dispara cada vez que cambiamos el estado del checkbox (tanto cuando lo seleccionamos como cuando lo deseleccionamos):

```
  function cambiocheckbox()  {
        alert(document.getElementById('checkbox1').checked);
```

```
    }
```

La función cambioradio se dispara cada vez que seleccionamos alguno de los tres radio:

```
  function cambioradio()  {
        alert(document.getElementById('radioa').checked+' '+
        document.getElementById('radiob').checked+' '+
        document.getElementById('radioc').checked);
    }
```

La función cambioselect se dispara cuando cambiamos de item del control select:

```
 function cambioselect() {
   alert(document.getElementById('select1').value);
 }
```

La función cambiotext y cambiotextarea se ejecuta cuando perdemos el foco del control y hemos producido un cambio en su contenido (agregado o borrado caracteres de su interior):

```
  function cambiotext() {
      alert(document.getElementById('text1').value);
    }

    function cambiotextarea() {
      alert(document.getElementById('textarea1').value);
    }
```

Eventos: focus y blur

El evento focus se dispara cuando se activa el control o toma foco y el evento blur se dispara cuando pierde el foco el control. Podemos capturar el evento focus y blur de un control de tipo text, textarea, button, checkbox, file, password, radio, reset y submit.

Problema: Confeccionar un formulario que muestre dos controles de tipo text. El que está con foco mostrar su texto de color rojo y aquel que no está seleccionado el texto se debe mostrar de color negro.

```
<html>
<head>

<script type="text/javascript">

  addEventListener('load',inicio,false);
```

```
  function inicio() {
    document.getElementById('text1').addEventListener('focus',tomarfoco1,false);
    document.getElementById('text2').addEventListener('focus',tomarfoco2,false);
    document.getElementById('text1').addEventListener('blur',perderfoco1,false);
    document.getElementById('text2').addEventListener('blur',perderfoco2,false);
  }

  function tomarfoco1() {
    document.getElementById('text1').style.color='#ff0000';
  }

  function tomarfoco2() {
    document.getElementById('text2').style.color='#ff0000';
  }

  function perderfoco1(){
    document.getElementById('text1').style.color='#000000';
  }

  function perderfoco2(){
    document.getElementById('text2').style.color='#000000';
  }

</script>

</head>
<body>

<input type="text" id="text1" name="text1" size="30">
<br>
<input type="text" id="text2" name="text2" size="30">

</body>
</html>
```

En la función inicio registramos los eventos fucus y blur para los dos controles text:

```
  function inicio()  {
       document.getElementById('text1').addEventListener('focus',tomarfoco1,false);
       document.getElementById('text2').addEventListener('focus',tomarfoco2,false);
       document.getElementById('text1').addEventListener('blur',perderfoco1,false);
       document.getElementById('text2').addEventListener('blur',perderfoco2,false);
     }
```

En los métodos tomarfoco1 y tomarfoco2 activamos el color rojo para el texto del control text:

```
  function tomarfoco1() {
       document.getElementById('text1').style.color='#ff0000';
     }
```

```
    function tomarfoco2() {
       document.getElementById('text2').style.color='#ff0000';
    }
```

En los métodos perderfoco1 y perderfoco2 procedemos a activar el color negro para los controles text:

```
  function perderfoco1() {
       document.getElementById('text1').style.color='#000000';
    }

    function perderfoco2() {
      document.getElementById('text2').style.color='#000000';
    }
```

Evento: submit

Todo formulario se le puede capturar el evento submit que se dispara previo a enviar los datos del formulario al servidor.

Uno de los usos más extendidos es la de validar los datos ingresados al formulario y abortar el envío de los mismos al servidos (con esto liberamos sobrecargas del servidor). El evento submit se dispara cuando presionamos un botón de tipo type="submit".

Para probar el funcionamiento del evento submit implementaremos un formulario que solicita la carga de una clave y la repetición de la misma. Luego cuando se presione un botón de tipo "submit" verificaremos que las dos claves ingresadas sean iguales.

```
<!doctype html>
<html>
<head>
<title></title>
<script type="text/javascript">

    window.addEventListener('load', inicio, false);

    function inicio() {
       document.getElementById("formulario1").addEventListener('submit', validar, false);
    }

    function validar(evt) {
       var cla1 = document.getElementById("clave1").value;
       var cla2 = document.getElementById("clave2").value;
       if (cla1!=cla2) {
```

```
            alert('Las claves ingresadas son distintas');
            evt.preventDefault();
        }
    }
</script>
</head>
<body>
<form method="post" action="procesar.php" id="formulario1">
Ingrese clave:
<input type="password" id="clave1" name="clave1" size="20" required>
<br>
Repita clave:
<input type="password" id="clave2" name="clave2" size="20" required>
<br>
<input type="submit" id="confirmar" name="confirmar" value="Confirmar">
</form>
</body>
</html>
```

Tengamos en cuenta que la primera línea indica que se trata de una página de HTML5:

```
<!doctype html>
```

Definimos un formulario que solicita la carga de dos claves y un botón submit para enviar los datos al servidor:

```
<form method="post" action="procesar.php" id="formulario1">
Ingrese clave:
<input type="password" id="clave1" name="clave1" size="20" required>
<br>
Repita clave:
<input type="password" id="clave2" name="clave2" size="20" required>
<br>
<input type="submit" id="confirmar" name="confirmar" value="Confirmar">
</form>
```

Registramos el evento load de la página indicando que se ejecute la función inicio donde registramos el evento submit del formulario:

```
window.addEventListener('load', inicio, false);

    function inicio() {
        document.getElementById("formulario1").addEventListener('submit', validar, false);
    }
```

La función validar extrae los contenidos de los dos "password" y verificamos si tienen string distintos en cuyo caso llamando al método preventDefault del

objeto que llega como parámetro lo cual previene que los datos se envíen al servidor:

```
function validar(evt) {
      var cla1 = document.getElementById("clave1").value;
      var cla2 = document.getElementById("clave2").value;
      if (cla1!=cla2) {
         alert('Las claves ingresadas son distintas');
         evt.preventDefault();
      }
   }
```

Función isNaN

La función global isNaN (is Not a Number) verifica si el valor que le pasamos es un número válido y podemos estar seguros de operar con dicho valor. Esta función puede ser empleada inmediatamente luego de llamar a las funciones parseInt y parseFloat.

Con un pequeño ejemplo podemos ver los valores que nos retorna la función isNaN si le pasamos variables con un valor entero, float, un string con caracteres numéricos y un string con caracteres alfabéticos:

```
var x=10;
if (isNaN(x))  //false
  alert('no es un número');
var z=10.5;
if (isNaN(z))  //false
  alert('no es un número');
var edad='77';
if (isNaN(edad))  //false
  alert('no es un número');
var nom='juan';
if (isNaN(nom))  //true
  alert('no es un número:'+nom);
```

Problema**:** Realizar la carga de dos valores enteros por teclado utilizando la función prompt. Calcular la suma y de forma previa controlar que los dos valores sean de tipo numérico.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var x1,x2,suma;
  x1=prompt('Ingrese el primer valor:','');
  x2=prompt('Ingrese el segundo valor:','');
  x1=parseInt(x1);
  x2=parseInt(x2);
```

```
  if (isNaN(x1) || isNaN(x2)){
    document.write('Al menos un valor ingresado no es numérico.');
  } else  {
    suma=x1+x2;
    document.write('La suma de los dos valores es:'+suma);
  }
</script>
</body>
</html>
```

Luego de cargar los dos valores por teclado procedemos a convertirlos a tipo entero:

```
  x1=prompt('Ingrese el primer valor:','');
    x2=prompt('Ingrese el segundo valor:','');
    x1=parseInt(x1);
    x2=parseInt(x2);
```

Mediante un if verificamos si alguno de los dos valores no es un número:

```
if (isNaN(x1) || isNaN(x2))  {
     document.write('Al menos uno de los dos valores ingresados no es numérico.');
}
```

Validación de formularios

La principal utilidad de JavaScript en el manejo de los formularios es la validación de los datos introducidos por los usuarios. Antes de enviar un formulario al servidor, se recomienda validar mediante JavaScript los datos insertados por el usuario. De esta forma, si el usuario ha cometido algún error al rellenar el formulario, se le puede notificar de forma instantánea, sin necesidad de esperar la respuesta del servidor.

Notificar los errores de forma inmediata mediante JavaScript mejora la satisfacción del usuario con la aplicación (lo que técnicamente se conoce como *"mejorar la experiencia de usuario"*) y ayuda a reducir la carga de procesamiento en el servidor.

Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario. En esta función, se comprueban si los valores que ha introducido el usuario cumplen las restricciones impuestas por la aplicación.

Aunque existen tantas posibles comprobaciones como elementos de formulario diferentes, algunas comprobaciones son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que la dirección de email indicada sea correcta, que la fecha

introducida sea lógica, que se haya introducido un número donde así se requiere, etc.

A continuación, se muestra el código JavaScript básico necesario para incorporar la validación a un formulario:

```
<form action="" method="" id="" name="" onsubmit="return validacion()">
  ...
</form>
```

Y el esquema de la función validacion() es el siguiente:

```
function validacion() {
  if (condicion que debe cumplir el primer campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  else if (condicion que debe cumplir el segundo campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  ...
  else if (condicion que debe cumplir el último campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }

  // Si el script ha llegado a este punto, todas las condiciones
  // se han cumplido, por lo que se devuelve el valor true
  return true;
}
```

El funcionamiento de esta técnica de validación se basa en el comportamiento del evento *onsubmit* de JavaScript. Al igual que otros eventos como *onclick* y *onkeypress*, el evento *'onsubmit* varía su comportamiento en función del valor que se devuelve.

Así, si el evento *onsubmit* devuelve el valor *true*, el formulario se envía como lo haría normalmente. Sin embargo, si el evento *onsubmit* devuelve el valor *false*, el formulario no se envía. La clave de esta técnica consiste en comprobar todos y cada uno de los elementos del formulario. En cuando se

encuentra un elemento incorrecto, se devuelve el valor false. Si no se encuentra ningún error, se devuelve el valor true.

Por lo tanto, en primer lugar se define el evento *onsubmit* del formulario como:

```
onsubmit="return validacion()"
```

Como el código JavaScript devuelve el valor resultante de la función validacion(), el formulario solamente se enviará al servidor si esa función devuelve true. En el caso de que la función validacion() devuelva false, el formulario permanecerá sin enviarse.

Dentro de la función validacion() se comprueban todas las condiciones impuestas por la aplicación. Cuando no se cumple una condición, se devuelve false y por tanto el formulario no se envía. Si se llega al final de la función, todas las condiciones se han cumplido correctamente, por lo que se devuelve true y el formulario se envía.

La notificación de los errores cometidos depende del diseño de cada aplicación. En el código del ejemplo anterior simplemente se muestran mensajes mediante la función alert() indicando el error producido. Las aplicaciones web mejor diseñadas muestran cada mensaje de error al lado del elemento de formulario correspondiente y también suelen mostrar un mensaje principal indicando que el formulario contiene errores.

Una vez definido el esquema de la función validacion(), se debe añadir a esta función el código correspondiente a todas las comprobaciones que se realizan sobre los elementos del formulario. A continuación, se muestran algunas de las validaciones más habituales de los campos de formulario.

- ## Validar un campo de texto obligatorio

Se trata de forzar al usuario a introducir un valor en un cuadro de texto o textarea en los que sea obligatorio. La condición en JavaScript se puede indicar como:

```
valor = document.getElementById("campo").value;
if( valor == null || valor.length == 0 || /^\s+$/.test(valor) ) {
  return false;
```

```
}
```

Para que se de por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que cero y que no se hayan introducido sólo espacios en blanco.

La palabra reservada null es un valor especial que se utiliza para indicar *"ningún valor"*. Si el valor de una variable es null, la variable no contiene ningún valor de tipo objeto, array, numérico, cadena de texto o booleano.

La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a cero caracteres, esto es, que no sea un texto vacío.

Por último, la tercera parte de la condición (/^\s+$/.test(valor)) obliga a que el valor introducido por el usuario no sólo esté formado por espacios en blanco. Esta comprobación se basa en el uso de *"expresiones regulares"*, un recurso habitual en cualquier lenguaje de programación pero que por su gran complejidad no se van a estudiar. Por lo tanto, sólo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.

- ## Validar un campo de texto con valores numéricos

Se trata de obligar al usuario a introducir un valor numérico en un cuadro de texto. La condición JavaScript consiste en:

```
valor = document.getElementById("campo").value;
if( isNaN(valor) ) {
  return false;
}
```

Si el contenido de la variable valor no es un número válido, no se cumple la condición. La ventaja de utilizar la función interna isNaN() es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

A continuación se muestran algunos resultados de la función isNaN():

```
isNaN(3);              // false
isNaN("3");            // false
isNaN(3.3545);         // false
```

```
isNaN(32323.345);       // false
isNaN(+23.2);           // false
isNaN("-23.2");         // false
isNaN("23a");           // true
isNaN("23.43.54");      // true
```

- **Validar que se ha seleccionado una opción de una lista**

Se trata de obligar al usuario a seleccionar un elemento de una lista desplegable. El siguiente código JavaScript permite conseguirlo:

```
indice = document.getElementById("opciones").selectedIndex;
if( indice == null || indice == 0 ) {
  return false;
}

<select id="opciones" name="opciones">
  <option value="">- Selecciona un valor -</option>
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
</select>
```

A partir de la propiedad *selectedIndex*, se comprueba si el índice de la opción seleccionada es válido y además es distinto de cero. La primera opción de la lista (- Selecciona un valor –) no es válida, por lo que no se permite el valor 0 para esta propiedad *selectedIndex*.

- **Validar una dirección de email**

Se trata de obligar al usuario a introducir una dirección de email con un formato válido. Por tanto, lo que se comprueba es que la dirección *parezca* válida, ya que no se comprueba si se trata de una cuenta de correo electrónico real y operativa. La condición JavaScript consiste en:

```
valor = document.getElementById("campo").value;
if( !(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)/.test(valor)) ) {
  return false;
}
```

La comprobación se realiza nuevamente mediante las expresiones regulares, ya que las direcciones de correo electrónico válidas pueden ser muy diferentes. Por otra parte, como el estándar que define el formato de las direcciones de correo electrónico es muy complejo, la expresión regular anterior es una simplificación. Aunque esta regla valida la mayoría de direcciones de correo electrónico utilizadas por los usuarios, no soporta todos los diferentes formatos válidos de email.

- ## Validar una fecha

Las fechas suelen ser los campos de formulario más complicados de validar por la multitud de formas diferentes en las que se pueden introducir. El siguiente código asume que de alguna forma se ha obtenido el año, el mes y el día introducidos por el usuario:

```
var ano = document.getElementById("ano").value;
var mes = document.getElementById("mes").value;
var dia = document.getElementById("dia").value;

valor = new Date(ano, mes, dia);

if( !isNaN(valor) ) {
  return false;
}
```

La función Date(ano, mes, dia) es una función interna de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. Es muy importante tener en cuenta que el número de mes se indica de 0 a 11, siendo 0 el mes de Enero y 11 el mes de Diciembre. Los días del mes siguen una numeración diferente, ya que el mínimo permitido es 1 y el máximo 31.

La validación consiste en intentar construir una fecha con los datos proporcionados por el usuario. Si los datos del usuario no son correctos, la fecha no se puede construir correctamente y por tanto la validación del formulario no será correcta.

- ## Validar un número de DNI

Se trata de comprobar que el número proporcionado por el usuario se corresponde con un número válido de Documento Nacional de Identidad o DNI. Aunque para cada país o región los requisitos del documento de identidad de las personas pueden variar, a continuación se muestra un ejemplo genérico fácilmente adaptable. La validación no sólo debe comprobar que el número esté formado por ocho cifras y una letra, sino que también es necesario comprobar que la letra indicada es correcta para el número introducido:

```
valor = document.getElementById("campo").value;
var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

if( !(/^\d{8}[A-Z]$/.test(valor)) ) {
  return false;
}

if(valor.charAt(8) != letras[(valor.substring(0, 8))%23]) {
  return false;
}
```

La primera comprobación asegura que el formato del número introducido es el correcto, es decir, que está formado por 8 números seguidos y una letra. Si la letra está al principio de los números, la comprobación sería /^[A-Z]\d{8}$/. Si en vez de ocho números y una letra, se requieren diez números y dos letras, la comprobación sería /^\d{10}[A-Z]{2}$/ y así sucesivamente.
La segunda comprobación aplica el algoritmo de cálculo de la letra del DNI y la compara con la letra proporcionada por el usuario. El algoritmo de cada documento de identificación es diferente, por lo que esta parte de la validación se debe adaptar convenientemente.

- ## Validar un número de teléfono

Los números de teléfono pueden ser indicados de formas muy diferentes: con prefijo nacional, con prefijo internacional, agrupado por pares, separando los números con guiones, etc.
El siguiente script considera que un número de teléfono está formado por nueve dígitos consecutivos y sin espacios ni guiones entre las cifras:

```
valor = document.getElementById("campo").value;

if( !(/^\d{9}$/.test(valor)) ) {
  return false;
}
```

Una vez más, la condición de JavaScript se basa en el uso de expresiones regulares, que comprueban si el valor indicado es una sucesión de nueve números consecutivos. A continuación, se muestran otras expresiones regulares que se pueden utilizar para otros formatos de número de teléfono:

Número	Expresión regular	Formato
900900900	/^\d{9}$/	9 cifras seguidas
900-900-900	/^\d{3}-\d{3}-\d{3}$/	9 cifras agrupadas de 3 en 3 y separadas por guiones
900 900900	/^\d{3}\s\d{6}$/	9 cifras, las 3 primeras separadas por un espacio
900 90 09 00	/^\d{3}\s\d{2}\s\d{2}\s\d{2}$/	9 cifras, las 3 primeras separadas por un espacio, las siguientes agrupadas de 2 en 2
(900) 900900	/^\(\d{3}\)\s\d{6}$/	9 cifras, las 3 primeras encerradas por paréntesis y un espacio de separación respecto del resto
+34 900900900	/^\+\d{2,3}\s\d{9}$/	Prefijo internacional (+ seguido de 2 o 3 cifras), espacio en blanco y 9 cifras consecutivas

- ## Validar que un checkbox ha sido seleccionado

Si un elemento de tipo *checkbox* se debe seleccionar de forma obligatoria, JavaScript permite comprobarlo de forma muy sencilla:

```
elemento = document.getElementById("campo");

if( !elemento.checked ) {
  return false;
}
```

Si se trata de comprobar que todos los *checkbox* del formulario han sido seleccionados, es más fácil utilizar un bucle:

```
formulario = document.getElementById("formulario");

for(var i=0; i<formulario.elements.length; i++) {
  var elemento = formulario.elements[i];
  if(elemento.type == "checkbox") {
    if(!elemento.checked) {
      return false;
    }
```

```
  }
}
```

- ## Validar que un radiobutton ha sido seleccionado

Aunque se trata de un caso similar al de los *checkbox*, la validación de los *radiobutton* presenta una diferencia importante: en general, la comprobación que se realiza es que el usuario haya seleccionado algún *radiobutton* de los que forman un determinado grupo. Mediante JavaScript, es sencillo determinar si se ha seleccionado algún *radiobutton* de un grupo:

```
opciones = document.getElementsByName("opciones");

var seleccionado = false;
for(var i=0; i<opciones.length; i++) {
  if(opciones[i].checked) {
    seleccionado = true;
    break;
  }
}

if(!seleccionado) {
  return false;
}
```

El anterior ejemplo recorre todos los *radiobutton* que forman un grupo y comprueba elemento por elemento si ha sido seleccionado. Cuando se encuentra el primer *radiobutton* seleccionado, se sale del bucle y se indica que al menos uno ha sido seleccionado.