MANUAL JAVASCRIPT

Qué es JavaScript?

JavaScript, al igual que Flash, Visual Basic Script, es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje HTML (lenguaje para el diseño de páginas de Internet). Al ser la más sencilla, es por el momento la más extendida. JavaScript no es un lenguaje de programación propiamente dicho como C, C++, Delphi, etc. Es un lenguaje script u orientado a documento, como pueden ser los lenguajes de macros que tienen muchos procesadores de texto y planillas de cálculo. No se puede desarrollar un programa con JavaScript que se ejecute fuera de un Navegador.
JavaScript es un lenguaje interpretado que se embebe en una página web HTML. Un lenguaje interpretado significa que a las instrucciones las analiza y procesa el navegador en el momento que deben ser ejecutadas.

Nuestro primer programa será el famoso "Hola Mundo", es decir un programa que muestre en el documento HTML el mensaje "Hola Mundo".

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

<script languaje="javascript">

document.write('Hola Mundo');

</script>



</body>

</html>
```

El programa en JavaScript debe ir encerrado entre la marca script e inicializada la propiedad languaje con la cadena javascript:

```
<script languaje="javascript">

</script>
```

Para imprimir caracteres sobre la página debemos llamar al comando 'write' del objeto document. La información a imprimirse debe ir entre comillas y encerrada entre paréntesis. Todo lo que indicamos entre comillas aparecerá tal cual dentro de la página HTML.
Es decir, si pedimos al navegador que ejecute esta página mostrará el texto 'Hola Mundo'.
Cada vez que escribimos una instrucción finalizamos con el caracter punto y coma.

ES IMPORTANTISIMO TENER EN CUENTA QUE JavaScript es SENSIBLE A MAYUSCULAS Y MINUSCULAS. NO ES LO MISMO ESCRIBIR:
document.write que DOCUMENT.WRITE (la primera forma es la correcta, la segunda forma provoca un error de sintaxis).

Nos acostumbraremos a prestar atención cada vez que escribamos en minúsculas o mayúsculas para no cometer errores sintácticos. Ya veremos que los nombres de funciones llevan letras en mayúsculas.

Variables.

Una variable es un depósito donde hay un valor. Consta de un nombre y pertenece a un tipo (númerico, cadena de caracteres, etc.).
Tipos de variable:
Una variable puede almacenar:
Valores Enteros (100, 260, etc.)
Valores Reales (1.24, 2.90, 5.00, etc.)
Cadenas de caracteres ("Juan", "Compras", "Listado", etc.)
Valores lógicos (true,false)
Existen otros tipos de variables que veremos más adelante.

Las variables son nombres que ponemos a los lugares donde almacenamos la información. En JavaScript, deben comenzar por una letra o un subrayado (_), pudiendo haber además dígitos entre los demás caracteres. Una variable no puede tener el mismo nombre de una palabra clave del lenguaje.
Una variable se define anteponiéndole la palabra clave var:
var dia;
se pueden declarar varias variables en una misma línea:
var dia, mes, anio;
a una variable se la puede definir e inmediatamente inicializarla con un valor:
var edad=20;
o en su defecto en dos pasos:
var edad;
edad=20;

Elección del nombre de una variable:
Debemos elegir nombres de variables representativos. En el ejemplo anterior los nombres dia, mes, anio son lo suficientemente claros para darnos una idea acabada sobre su contenido, una mala elección de nombres hubiera sido llamarlas a,b y c. Podemos darle otros buenos nombres. Otros no son tan representativos, por ejemplo d, m, a. Posiblemente cuando estemos resolviendo un problema dicho nombre nos recuerde que almacenamos el dia, pero pasado un tiempo lo olvidaríamos.

Impresión de variables en una página HTML.
Para mostrar el contenido de una variable en una página utilizamos el objeto document y llamamos a la función write.
En el siguiente ejemplo definimos una serie de variables y las mostramos en la página:

```
<html>
<head>
</head>
<body>
<script languaje="JavaScript">
var nombre='Juan';
var edad=10;
var altura=1.92;
var casado=false;
document.write(nombre);
document.write('<br>');
document.write(edad);
document.write('<br>');
document.write(altura);
```

```
  document.write('<br>');
  document.write(casado);
  </script>
  </body>
  </html>
```
Cuando imprimimos una variable, no la debemos disponer entre simples comillas (en caso de hacer esto, aparecerá el nombre de la variable y no su contenido)
Los valores de las variables que almacenan nombres (es decir, son cadenas de caracteres) deben ir encerradas entre comillas simples. Los valores de las variables enteras (en este ejemplo la variable edad) y reales no deben ir encerradas entre comillas. Cada instrucción finaliza con un punto y coma.
Las variables de tipo boolean pueden almacenar solo dos valores: true o false.
El resultado al visualizar la página debe ser 4 líneas similares a éstas:

Juan
10
1.92
false

Es decir que se muestran los contenidos de las 4 variables. Una variable es de un tipo determinado cuando le asignamos un valor:
```
  var edad=10;
```
Es de tipo entera ya que le asignamos un valor entero.
```
  var nombre='juan';
```
Es de tipo cadena.
Para mostrar el contenido de una variable en una página debemos utilizar la función 'write' que pertenece al objeto document. Recordemos que el lenguaje JavaScript es sensible a mayúsculas y minúsculas y no será lo mismo si tipeamos:
Document.Write(nombre);
Esto porque no existe el objeto 'Document' sino el objeto 'document' (con d minúscula), lo mismo no existe la función 'Write' sino 'write', este es un error muy común cuando comenzamos a programar en JavaScript

Entrada de datos por teclado.

Para la entrada de datos por teclado tenemos la función prompt. Cada vez que necesitamos ingresar un dato con esta función, aparece una ventana donde cargamos el valor. Hay otras formas más sofisticadas para la entrada de datos en una página HTML, pero para el aprendizaje de los conceptos básicos de JavaScript nos resultará más práctica esta función.
Para ver su funcionamiento analicemos este ejemplo:

```
<html>
  <head>
  </head>
  <body>
  <script languaje="JavaScript">
  var nombre;
  var edad;
  nombre=prompt('Ingrese su nombre:','');
  edad=prompt('Ingrese su edad:','');
  document.write('Hola ');
  document.write(nombre);
  document.write(' asi que tienes ');
  document.write(edad);
  document.write(' años');
  </script>
  </body>
```

```
  </html>
```
La sintaxis de la función prompt es:
```
<variable que recibe el dato>=prompt(<mensaje a mostrar en la
ventana>,<valor
  inicial a mostrar en la ventana>);
```
La función prompt tiene dos parámetros: uno es el mensaje y el otro el valor incial a mostrar

Estructuras secuenciales de programación.

Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina estructura secuencial.
El problema anterior, donde se ingresa el nombre de una persona y su edad se trata de una estructura secuencial.
Ejemplo de otro algoritmo con estructura secuencial: Realizar la carga de dos números por teclado e imprimir su suma y su producto:

```
<html>
  <head>
  <script languaje="JavaScript">
  var valor1;
  var valor2;
  valor1=prompt('Ingrese primer número:','');
  valor2=prompt('Ingrese segundo número','');
  var suma=parseInt(valor1)+parseInt(valor2);
  var producto=valor1*valor2;
  document.write('La suma es ');
  document.write(suma);
  document.write('<br>');
  document.write('El producto es ');
  document.write(producto);
  </script>
  </head>
  <body>
  </body>
  </html>
```
Lo primero que debemos tener en cuenta es que si queremos que el operador + sume los contenidos de los valores numéricos ingresados por teclado, debemos llamar a la función parseInt y pasarle como parámetro las variables valor1 y valor2 sucesivamente. Con esto logramos que el operador más, sume las variables como enteros y no como cadenas de caracteres. Si por ejemplo sumamos 1 + 1 sin utilizar la función parseInt el resultado será 11 en lugar de 2, ya que el operador + concatena las dos cadenas.
Cuando empleamos el operador * para el producto, ya no es obligatorio utilizar la función parseInt (es decir, sólo para el operador + debemos utilizarla).
En JavaScript, como no podemos indicarle de qué tipo es la variable, requiere mucho más cuidado cuando operamos con sus contenidos.
Este problema es secuencial ya que ingresamos dos valores por teclado, luego hacemos dos operaciones y por último mostramos los resultados.

Estructuras condicionales simples.

No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando hay que tomar una decisión aparecen las estructuras condicionales.
En nuestra vida diaria se nos presentan situaciones donde debemos decidir.
¿Elijo la carrera A o la carrera B ?
¿Me pongo este pantalón ?

¿Entro al sitio A o al sitio B ?
Para ir al trabajo, ¿elijo el camino A o el camino B ?
Al cursar una carrera, ¿elijo el turno mañana, tarde o noche ?

Por supuesto que en un problema se combinan estructuras secuenciales y condicionales.
Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizarla.
En una estructura CONDICIONAL SIMPLE por el camino del verdadero hay actividades y por el camino del falso no hay actividades. Por el camino del verdadero pueden existir varias operaciones, entradas y salidas, inclusive ya veremos que puede haber otras estructuras condicionales.
Ejemplo: Realizar la carga de una nota de un alumno. Mostrar un mensaje que aprobó si tiene una nota mayor o igual a 4:

```
<html>
<head>
</head>
<body>
<script languaje="javascript">
var nombre;
var nota;
nombre=prompt('Ingrese nombre:','');
nota=prompt('Ingrese su nota:','');
if (nota>=4)
{
  document.write(nombre+' esta aprobado con un '+nota);
}
</script>
</body>
</html>
```

Aparece la instrucción if en el lenguaje JavaScript. La condición debe ir entre paréntesis. Si la condición se verifica verdadera se ejecuta todas las instrucciones que se encuentran encerradas entre las llaves de apertura y cerrado seguidas al if.
Para disponer condiciones en un if podemos utilizar alguno de los siguientes operadores relacionales:

```
>  mayor
>= mayor o igual
<  menor
<= menor o igual
!= distinto
== igual
```

Siempre debemos tener en cuenta que en la condición del if deben intervenir una variable un operador relacional y otra variable o valor fijo.
Otra cosa que hemos incorporado es el operador + para cadenas de caracteres:

```
document.write(nombre+' esta aprobado con un '+nota);
```

Con esto hacemos más corto la cantidad de líneas de nuestro programa, recordemos que veníamos haciéndolo de la siguiente forma:

```
document.write(nombre);
document.write(' esta aprobado con un ');
document.write(nota);
```

Estructuras condicionales anidadas.

Decimos que una estructura condicional es anidada cuando por la rama del verdadero o el falso de una estructura condicional hay otra estructura condicional.
Ejemplo: Confeccionar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima alguno de estos mensajes:
Si el promedio es >=7 mostrar "Promocionado".
Si el promedio es >=4 y <7 mostrar "Regular".
Si el promedio es <4 mostrar "Reprobado".
Solución:

```
<html>
<head>
</head>
<body>
<script languaje="javascript">
var nota1,nota2,nota3;
nota1=prompt('Ingrese 1ra. nota:','');
nota2=prompt('Ingrese 2da. nota:','');
nota3=prompt('Ingrese 3ra. nota:','');
//Convertimos los 3 string en enteros
nota1=parseInt(nota1);
nota2=parseInt(nota2);
nota3=parseInt(nota3);
var pro;
pro=(nota1+nota2+nota3)/3;
if (pro>=7)
{
  document.write('promocionado');
}
else
{
  if (pro>=4)
  {
    document.write('regular');
  }
  else
  {
    document.write('reprobado');
  }
}
</script>
</body>
</html>
```

Analicemos el siguiente programa. Se ingresan tres string por teclado que representan las notas de un alumno, se transformas a variables enteras y se obtiene el promedio sumando los tres valores y dividiendo por 3 dicho resultado. Primeramente preguntamos si el promedio es superior o igual a 7, en caso afirmativo por la rama del verdadero de la estructura condicional mostramos un mensaje que indique 'Promocionado' (con comillas indicamos un texto que debe imprimirse en pantalla).
En caso que la condición nos de falso, por la rama del falso aparece otra estructura condicional, porque todavía debemos averiguar si el promedio del alumno es superior/ igual a cuatro o inferior a cuatro.

Los comentarios en JavaScript los hacemos disponiendo dos barras previas al comentario:

```
//Convertimos los 3 string en enteros
```

Si queremos disponer varias líneas de comentarios tenemos como alternativa:

```
/*
```

```
linea de comentario 1.
linea de comentario 2.
etc.
*/
```
Es decir encerramos el bloque con los caracteres /* */

Operadores lógicos && (y) en las estructuras condicionales.

El operador &&, traducido se lo lee como "Y". Se emplea cuando en una estructura condicional se disponen dos condiciones.
Cuando vinculamos dos o más condiciones con el operador "&&" las dos condiciones deben ser verdaderas para que el resultado de la condición compuesta de Verdadero y continúe por la rama del verdadero de la estructura condicional.
Recordemos que la condición debe ir entre paréntesis en forma obligatoria.
La utilización de operadores lógicos permiten en muchos casos, plantear algoritmos más cortos y comprensibles.
Veamos un ejemplo: Confeccionar un programa que lea por teclado tres números distintos y nos muestre el mayor de ellos.

```
<html>
<head>
</head>
<body>
<script languaje="javascript">
var num1,num2,num3;
num1=prompt('Ingrese primer número:','');
num2=prompt('Ingrese segundo número:','');
num3=prompt('Ingrese tercer número:','');
num1=parseInt(num1);
num2=parseInt(num2);
num3=parseInt(num3);
if (num1>num2 && num1>num3)
{
  document.write('el mayor es el '+num1);
}
else
{
  if (num2>num3)
  {
    document.write('el mayor es el '+num2);
  }
  else
  {
    document.write('el mayor es el '+num3);
  }
}
</script>
</body>
</html>
```
Podemos leerla de la siguiente forma:
Si el contenido de la variable num1 es mayor al contenido de la variable num2 Y si el contenido de la variable num1 es mayor al contenido de la variable num3 entonces la CONDICION COMPUESTA resulta Verdadera.
Si una de las condiciones simples da falso, la CONDICION COMPUESTA da Falso y continúa por la rama del falso.
Es decir que se mostrará el contenido de num1 si y sólo si num1>num2 y num1>num3.

En caso de ser Falsa la condición de la rama del falso, analizamos el contenido de num2 y num3 para ver cual tiene un valor mayor.
En esta segunda estructura condicional, al haber una condición simple, no se requieren operadores lógicos

Operadores lógicos || (o) en las estructuras condicionales.

Traducido se lo lee como "O". Si la condición 1 es Verdadera o la condición 2 es Verdadera, luego ejecutar la rama del Verdadero.
Cuando vinculamos dos o más condiciones con el operador "O", con que una de las dos condiciones sea Verdadera alcanza para que el resultado de la condición compuesta sea Verdadero.

Ejemplo: Se carga una fecha (día, mes y año) por teclado. Mostrar un mensaje si corresponde al primer trimestre del año (enero, febrero o marzo).
Cargar por teclado el valor numérico del día, mes y año por separado.

```
<html>
<head>
</head>
<body>
<script languaje="javascript">
var dia,mes,año;
dia=prompt('Ingrese día:','');
mes=prompt('Ingrese mes:','');
año=prompt('Ingrese año:','');
dia=parseInt(dia);
mes=parseInt(mes);
año=parseInt(año);
if (mes==1 || mes==2 || mes==3)
{
  document.write('corresponde al primer trimestre del año.');
}
</script>
</body>
</html>
```

La carga de una fecha se hace por partes, ingresamos las variables dia, mes y año

Estructuras switch.

La instrucción switch es una alternativa para remplazar los if/else if.
De todos modos se puede aplicar en ciertas situaciones donde la condición se verifica si es igual a cierto valor. No podemos preguntar por mayor o menor.
Con un ejemplo sencillo veremos cual es su sintaxis. Confeccionar un programa que solicite que ingrese un valor entre 1 y 5. Luego mostrar en castellano el valor ingresado. Mostrar un mensaje de error en caso de haber ingresado un valor que no se encuentre en dicho rango.

```
<html>
<head>
</head>
<body>
<script languaje="javascript">
var valor;
valor=prompt('Ingrese un valor comprendido entre 1 y 5:','');
//Convertimos a entero
```

```
valor=parseInt(valor);
switch (valor) {
  case 1: document.write('uno');
          break;
  case 2: document.write('dos');
          break;
  case 3: document.write('tres');
          break;
  case 4: document.write('cuatro');
          break;
  case 5: document.write('cinco');
          break;
  default:document.write('debe ingresar un valor comprendido entre 1 y
5.');
}
</script>
<A href="pagina2.html">Ver segundo problema</a>
</body>
</html>
```

Debemos tener en cuenta que la variable que analizamos debe ir después de la instrucción switch entre paréntesis. Cada valor que se analiza debe ir luego de la palabra clave 'case' y seguido a los dos puntos, las instrucciones a ejecutar, en caso de verificar dicho valor la variable que analiza el switch.
Es importante disponer la palabra clave 'break' al finalizar cada caso. La instrucciones que hay después de la palabra clave 'default' se ejecutan en caso que la variable no se verifique en algún case. De todos modos el default es opcional en esta instrucción.

Plantearemos un segundo problema para ver que podemos utilizar variables de tipo cadena con la instrucción switch.
Ingresar por teclado el nombre de un color (rojo, verde o azul), luego pintar el fondo de la ventana con dicho color:

```
<html>
<head>
</head>
<body>
<script languaje="javascript">
var col;
col=prompt('Ingrese el color con que se quiere pintar el fondo de la
ventana
           (rojo, verde, azul)' ,'');
switch (col) {
  case 'rojo': document.bgColor='#ff0000';
               break;
  case 'verde': document.bgColor='#00ff00';
               break;
  case 'azul': document.bgColor='#0000ff';
               break;
}
</script>
</body>
</html>
```

Cuando verificamos cadenas debemos encerrarlas entre comillas el valor a analizar:

```
  case 'rojo': document.bgColor='#ff0000';
               break;
```

Para cambiar el color de fondo de la ventana debemos asignarle a la propiedad bgColor del objeto document el color a asignar (el color está formado por tres valores hexadecimales que representan la cantidad de rojo, verde y azul), en este caso al valor de rojo le asignamos ff (255 en decimal) es decir el valor máximo posible, luego 00 para verde y azul (podemos utilizar algún software de graficación para que nos genere los tres valores).

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. En caso que la condición sea Falsa continua 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.

```
<html>
<head>
</head>
<body>
<script languaje="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).

Concepto de acumulador.

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

```
<html>
<head>
</head>
<body>
<script languaje="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 los valores ingresados.

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 languaje="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.

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 languaje="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 avisara.

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 como 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 languaje="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 languaje="javascript">
function mostrarMensaje()
{
  document.write("Cuidado<br>");
  document.write("Ingrese su documento correctamente<br>");
}



mostrarMensaje();

mostrarMensaje();

mostrarMensaje();



</script>

</body>

</html>
```

Recordemos que JavaScript es sencible 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 languaje="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. Luego de cargar los dos valores por teclado se llama a la función mostrarComprendidos y le enviamos las variables valor1 y valor2. Los parámetors x1 y x2 reciben los contenidos de las variables valor1 y valor 2.
Es importante notar que a la función la podemos llamar la cantidad de veces que la necesitemos

Funciones que retornan un valor.

Son comunes los casos donde una función, luego de hacer un proceso, retorne un valor.
Ejemplo 1: 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 languaje="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;

valor=prompt("Ingrese un valor entre 1 y 5","");

valor=parseInt(valor);

var r;

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.

Ejemplo 2: 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 2003".

```
<html>
<head>
</head>
<body>
<script languaje="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(11,6,2006));



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

Analicemos un poco la función formatearFecha. Llegan tres parámetros con el día, mes y año. Definimos e inicializamos una variable con:

```
    var s='Hoy es '+dia+' de ';
```
Luego le concatenamos o sumamos el mes:
```
s=s+'enero ';
```
Esto, si el parámetro mes tiene un uno. Observemos como acumulamos lo que tiene 's' más el string 'enero '. En caso de hacer s='enero ' perderíamos el valor previo que tenía la variable s. Por último concatenamos el año:

```
s=s+'de '+año;
```
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(11,6,2006);
  document.write(fec);
```
Guardamos en la variable 'fec' el string devuelto por la función

Clase Date

JavaScript dispone de varias clases predefinidos 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:

```
fecha = new Date();//creación de un objeto de la clase 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 nuestra computadora. En caso contrario hay que tener en cuenta que los meses comienzan por cero. Así, por ejemplo:
```
navidad06 = new Date(2006, 11, 25)
```
El objeto Date dispone, entre otros, de los siguientes métodos:
```
  getYear()
  setYear(año)
    Obtiene y coloca, respectivamente, el año de la fecha.
    Éste se devuelve como número de 4 dígitos excepto en el
    caso en que esté entre 1900 y 1999, en cuyo caso
    devolverá las dos últimas cifras.
  getFullYear()
  setFullYear(año)
    Realizan la misma función que los anteriores, pero sin
    tanta complicación, ya que siempre devuelven números
    con todos sus dígitos.
  getMonth()
  setMonth(mes)
  getDate()
  setDate(dia)
  getHours()
  setHours(horas)
  getMinutes()
  setMinutes(minutos)
  getSeconds()
  setSeconds(segundos)
    Obtienen y colocan, respectivamente, el mes, día, hora,
    minuto y segundo 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)
```
Ejemplo: Mostrar en una página la fecha y la hora actual.
```
<HTML>
<HEAD>
```

```
<SCRIPT LANGUAGE="JavaScript">
function mostrarFechaHora()
{
  var fecha
  fecha=new Date();
  document.write('Hoy es ');
  document.write(fecha.getDate()+'/');
  document.write((fecha.getMonth()+1)+'/');
  document.write(fecha.getYear());
  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>
```

En este problema hemos creado un objeto de la clase Date. Luego llamamos una serie de métodos que nos retornan datos sobre la fecha y hora actual del equipo de computación donde se está ejecutando el navegador.
Es bueno notar que para llamar a los métodos disponemos:
<nombre de objeto>.<nombre de método>(parámetros)

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 elemento del mismo (componente).

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 LANGUAGE="JavaScript">
function cargar(sueldos)
{
  var f;
  for(f=0;f<sueldos.length;f++)
  {
    var v;
    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 la 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 LANGUAGE="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 String.

Un string consiste en uno o más caracteres encerrados entre simple o doble comillas.
Concatenación de cadenas (+)
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.

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

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

Métodos
charAt(pos)
Retorna el caracter 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 caracter '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 caracter.

```
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();
```

Luego 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();
```

Luego 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 languaje="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 caracter 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 en problema muy sencillo empleando el lenguaje JavaScript; dispondremos un botón y cada vez que se presione, mostraremos un contador:

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

<script languaje="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 NAME 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 languaje="JavaScript">
function mostrar()
{
  var nom=document.form1.nombre.value;
  var ed=document.form1.edad.value;
  alert('Ingreso el nombre:' + nom);
  alert('Y la edad:' + ed);
}
</script>

<form name="form1">
  Ingrese su nombre:
  <input type="text" name="nombre"><br>
  Ingrese su edad:
  <input type="text" name="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.form1.nombre.value;
  var ed=document.form1.edad.value;
```

Para hacer más clara la función guardamos en dos variables auxiliares los contenidos de los controles de tipo TEXT. Hay que tener en cuenta que a nuestra página la accedemos por medio del objeto: document, luego, al formulario que hemos creado, lo accedemos por el NAME que le dimos al formulario, en este caso: form1, luego, a cada control que contiene el formulario, lo accedemos nuevamento por su NAME, es decir: nombre y edad respectivamente. Por último, para acceder a las cadenas cargadas debemos indicar la propiedad value.

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 tipeamos.

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>
<SCRIPT LANGUAGE="JavaScript">
function validar()
{
  if (document.form1.text1.value.length<5)
  {
    alert("Ingrese al menos 5 caracteres");
    document.form1.text1.value="";
  }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="form1">
Ingrese clave(al menos 5 caracteres): <INPUT TYPE="password"
NAME="text1">
<INPUT TYPE="button" ONCLICK="validar()" VALUE="Enviar">
</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 a:

```
document.form1.text1.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 (document.form1.text1.value.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.form1.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.form1.select1.options[document.form1.select1.selectedIndex].t
ext
```

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.form1.select1.options[document.form1.select1.selectedIndex].v
alue
```

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

```
<html>
<head>
</head>
<body>
<script languaje="JavaScript">
function cambiarColor()
{
  document.form1.text1.value = document.form1.select1.selectedIndex;
  document.form1.text2.value =

document.form1.select1.options[document.form1.select1.selectedIndex].t
ext;
  document.form1.text3.value =
     document.form1.select1.options
[document.form1.select1.selectedIndex].value;
}
</script>
<form name="form1">
<select size="1" name="select1" ONCHANGE="cambiarColor()">
<option value="0xff0000">Rojo</option>
<option value="0x00ff00">Verde</option>
<option value="0x0000ff">Azul</option>
</select></p>
<br>
Número de índice seleccionado del objeto SELECT:<input type="text"
name="text1"><br>
Texto seleccionado:<input type="text" name="text2"><br>
Valor asociado:<input type="text" name="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:

```
Rojo
```

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 como podemos acceder a su estado desde JavaScript haremos un 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 languaje="JavaScript">
function contarSeleccionados()
{
  var cant=0;
  if (document.form1.lenguaje1.checked)
    cant++;
  if (document.form1.lenguaje2.checked)
    cant++;
  if (document.form1.lenguaje3.checked)
    cant++;
  if (document.form1.lenguaje4.checked)
    cant++;
  alert('Conoce ' + cant + ' lenguajes');
}
</script>
<form name="form1">
<input type="checkbox" name="lenguaje1">JavaScript
<br>
<input type="checkbox" name="lenguaje2">PHP
<br>
<input type="checkbox" name="lenguaje3">JSP
<br>
<input type="checkbox" name="lenguaje4">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 que almacena true o false según esté o no seleccionado el control.
Disponemos un 'if' para cada checkbox:

```
  if (document.form1.lenguaje1.checked)
    cant++;
```

Control RADIO

Los objetos RADIO tienen sentido cuando disponemos varios elementos. Sólo 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 languaje="JavaScript">
function mostrarSeleccionado()
{
  if (document.form1.estudios[0].checked)
    alert('no tienes estudios');
  if (document.form1.estudios[1].checked)
    alert('tienes estudios primarios');
  if (document.form1.estudios[2].checked)
    alert('tienes estudios secundarios');
  if (document.form1.estudios[3].checked)
    alert('tienes estudios universitarios');
}
</script>
<form name="form1">
<input type="radio" name="estudios" value="sin estudios">Sin
estudios
<br>
<input type="radio" name="estudios">Primarios
<br>
<input type="radio" name="estudios">Secundarios
<br>
<input type="radio" 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 el mismo name. Luego podemos acceder a cada elemento por medio de un subíndice:

```
  if (document.form1.estudios[0].checked)
    alert('no tienes estudios');
```

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

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 languaje="JavaScript">
function controlarCaracteres()
{
  if (document.form1.curriculum.value.length>2000)
    alert('curriculum muy largo');
else
alert('datos correctos');
}
</script>
<form name="form1">
&lt;textarea name=&quot;curriculum&quot; rows="10" cols="50"
&gt;&lt;/textarea&gt;
<br>
<input type="button" value="Mostrar" onClick="controlarCaracteres()">
</form>
</body>
</html>
```

Para saber el largo de la cadena cargada:

```
  if (document.form1.curriculum.value.length>2000)
```

accedemos a la propiedad length

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.

```
<html>
<head></head>
<body>
<script languaje="JavaScript">
function vaciar(control)
{
  control.value='';
}
function verificarEntrada(control)
{
  if (control.value=='')
    alert('Debe ingresar datos');
}
</script>
<form name="form1">
Ingrese su nombre:
<input type="text" name="nombre" onFocus="vaciar(this)"
onBlur="verificarEntrada(this)"><br>
Ingrese su edad:
<input type="text" name="edad" onFocus="vaciar(this)"
onBlur="verificarEntrada(this)"><br>
<input type="button" value="Confirmar">
</form>
</body>
</html>
```

A cada control de tipo TEXT le inicializamos los eventos onFocus y onBlur. 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. 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

Tener en cuenta que para que el navegador IExplorer muestre los mensajes de error debemos ir a Herramientas --> Opciones de Internet --> pestaña "Opciones Avanzadas" y en la sección "Examinar" desmarcar la casilla de "Deshabilitar la depuración de secuencia de comandos"...) Con Mozilla FireFox podemos ver los errores luego de producidos mediante Herramientas --> Consola JavaScript.)

Eventos onMouseOver y onMouseOut

El evento onMouseOver se ejecuta cuando pasamos la flecha del mouse sobre un hipervínculo 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 languaje="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>
<br>
<br>
<br>
<a href="pagina2.html">ver segundo problema</a>
</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;
}
```

El segundo problema pinta de color el interior de una casilla de una tabla y lo regresa a su color original cuando salimos de la misma:

```
<html>
<head></head>
<body>
<script languaje="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>
```

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 languaje="JavaScript">
function activarPrimerControl()
{
  document.form1.nombre.focus();
}
</script>
<form name="form1">
Ingrese su nombre:
<input type="text" name="nombre"><br>
Ingrese su edad:
<input type="text" name="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.form1.nombre.focus();
```

```
}
```

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
       (a esta responsabilidad 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.

```
Ej:
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.

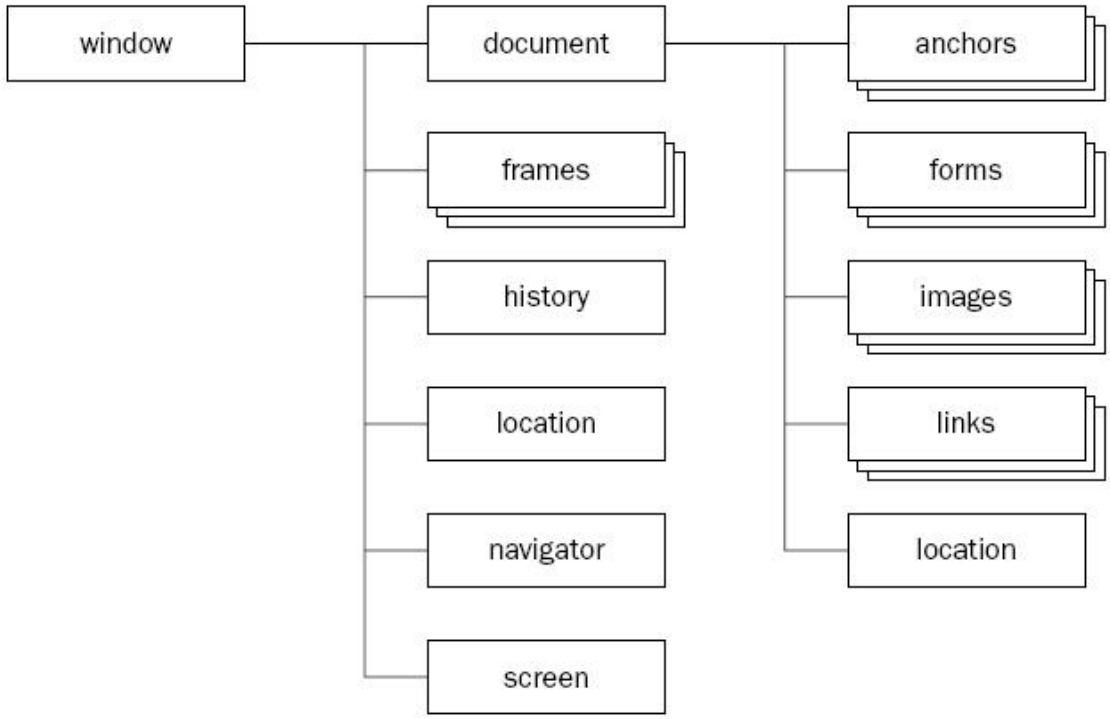
El siguiente programa muestra varios de los métodos disponibles del objeto window:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function cerrar()
{
  close(); // podemos escribir window.close();
}

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

function abrirParametros()
{
  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>
</HEAD>
<BODY>
Este programa permite analizar la llamada a distintas
responsabilidades del
objeto window.<br>
<FORM>
<input type="button" value="close()" onClick="cerrar()">

<br><br>
<input type="button" value="open()" onClick="abrir()">
<br><br>
<input type="button" value="open con parámetros"
onClick="abrirParametros()" >
<br><br>
<input type="button" value="alert" onClick="mostrarAlerta()">
<br><br>
<input type="button" value="confirm" onClick="confirmar()">
```

```
<br><br>
<input type="button" value="prompt" onClick="cargarCadena()">
</FORM>
</BODY>
</HTML>
```

Propiedad location del objeto window

El objeto location colabora directamente con el objeto window:



Cuando le asignamos una nueva dirección a la propiedad location del objeto window, el navegador redirecciona a dicha página.
Implementaremos un pequeño ejemplo para ver la utilidad de esta propiedad:
Supongamos que tenemos un hipervínculo que al ser presionado muestre una vetana de confirmación, si queremos ingresar a un sitio para mayores. En caso que el visitante presione el botón afirmativo, redireccionamos a otra página, en caso contrario mostramos un mensaje:

```
<html>
<head>
</head>
<body>
<script languaje="javascript">
function verificarMayorEdad()
{
  if (window.confirm('Es mayor de edad?'))
    window.location='pagina2.html';
  else
    window.alert('Cuando sea mayor de edad podrá ingresar');
}
```

```
</script>
<a href="javascript:verificarMayorEdad()">Ingresar al sitio para
mayores</a>
</body>
</html>
```

Lo primero que tenemos que indicar es que para llamar a una función de javascript desde un hipervínculo debemos anteceder la palabra javascript seguida de dos puntos y por último, el nombre de la función:

```
<a href="javascript:verificarMayorEdad()">
```

La función verificarMayorEdad muestra la ventana con los botones confirmar y cancelar (recordar que el método confirm del objeto window hace esto en forma automática).
Si se presiona el botón confirmar, la función confirm retorna true y por lo tanto se ejecuta el verdadero del if:

```
  if (window.confirm('Es mayor de edad?'))
    window.location='pagina2.html';
  else
    window.alert('Cuando sea mayor de edad podrá ingresar');
```

Recordar que anteceder la palabra window a estas funciones y propiedades es opcional.
Por último la página que se redirecciona es:

```
<html>
<head>
<title>Problema</title>
</head>
<body>
Bienvenido al sitio para adultos.
</body>
</html>
```

Propiedad history del objeto window

El objeto history colabora directamente con el objeto window:

El objeto history almacena todas las páginas que visitamos. Luego, con una serie de funciones, podemos extraer de la memoria de la computadora las páginas ya visitadas, sin tener que pedirlas nuevamente al servidor.
Cuenta con las siguientes funciones:

```
  window.history.back();    //Retrocede a la página anterior
  window.history.forward(); //Avanza a la página siguiente almacenada
en la
                            cache de la máquina.
  window.history.go();  //Avanza o retrocede en la lista de páginas
visitadas.
```

Llamar a la función back, tiene el mismo comportamiento que presionar el botón "Atrás" del navegador.
El siguiente ejemplo nos permite cargar una segunda página y luego retroceder a la primera página sin tener que solicitarla nuevamente al servidor:

```
<html>
<head>
<title>Problema</title>
<script languaje="javascript">
function avanzar()
{
  window.history.go(1);
}
</script>
</head>
<body>
<a href="pagina2.html">Ir a la página 2</a>
<br>
<br>
<a href="javascript:avanzar()">Extraer del cache la segunda página</a>
</body>
</html>
```

En esta primera página, la primera vez, debemos cargar la segunda página seleccionando el hipervínculo pagina2.
La segunda página:

```
<html>
<head>
<title>Problema</title>
<script languje="javascript">
function retornar()
{
  window.history.go(-1);
}
</script>
</head>
<body>
<a href="javascript:retornar()">Retornar</a>
</body>
</html>
```

En la segunda página, mediante el método go y pasándole un valor negativo, retrocedemos a la primera página sin tener la necesidad de recargarla.
Podemos mejorar el ejemplo accediendo al atributo length (almacena la cantidad de páginas de la lista) del objeto history:

```
if (window.history.length>0)
  window.history.go(1);
else
  alert('no hay otra página en la cache hacia adelante');
```

Propiedad screen del objeto window

El objeto screen colabora directamente con el objeto window:

El objeto screen ofrece información acerca del monitor donde se está ejecutando el navegador.
La propiedades principales del objeto screen son:

```
availHeight : El alto de la pantalla en pixeles disponible para el navegador.
availWidth : El ancho de la pantalla en pixeles disponible para el navegador.
colorDepth : Representa el número de bits usados para representar los colores.
height : El alto de la pantalla en pixeles.
width : El ancho de la pantalla en pixeles.
```

El siguiente programa muestra el valor almacenado en las cinco propiedades que tiene el objeto screen:

```
<html>
<head>
<title>Problema</title>
</head>
<body>
<script languaje="javascript">
document.write('Valores de las propiedades del objeto screen:<br>');
document.write('availHeight :' + screen.availHeight + '<br>');
document.write('availWidth :' + screen.availWidth + '<br>');
document.write('height :' + screen.height + '<br>');
document.write('width :' + screen.width + '<br>');
document.write('colorDepth :' + screen.colorDepth);
</script>
</body>
</html>
```

No olvidar que el objeto screen es una propiedad del objeto window, por lo que haber dispuesto la sintaxis: window.screen.width etc. es la forma más completa, pero más tediosa de escribir (recordar que el objeto window es el principal y lo podemos obviar cuando accedemos a sus propiedades o métodos).

Propiedad navigator del objeto window

Contiene información sobre el navegador web. La implementación de este objeto varía entre navegadores (IExplorer, FireFox, Opera, etc.)
Las propiedades comunes a estos navegadores son:

```
appName : almacena el nombre oficial del navegador.
appVersion : almacena la versión del navegador.
cookieEnabled : almacena si las cookies están activas en el navegador.
platform : almacena la plataforma donde el navegador se está
ejecutando.
plugins : almacena un array de los plugin cargados en el navegador.
```

Este pequeño programa muestra los valores de las propiedades antes anunciadas:

```
<html>
<head>
<title>Problema</title>
</head>
<body>
<script languaje="javascript">
document.write('Valores de las propiedades del objeto
navigator:<br>');
document.write('appName :' + navigator.appName + '<br>');
document.write('appVersion :' + navigator.appVersion + '<br>');
document.write('cookieEnabled :' + navigator.cookieEnabled + '<br>');
document.write('plugins :' + navigator.plugins.length + '<br>');
</script>
</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 tipear 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 language="javascript" type="text/javascript"
src="pagina2.js"></script>
</head>
<body>
<script languaje="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 language="javascript" type="text/javascript"
src="pagina2.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.

Programación orientada a objetos en JavaScript.

El lenguaje JavaScript no es un lenguaje orientado a objetos completo, pero permite definir clases con sus atributos y responsabilidades. Finalmente nos permite definir objetos de estas clases.
Pero el otro pilar de la programación orientada a objetos, es decir la herencia, no está implementada en el lenguaje.
Veremos la sintaxis para la declaración de una clase y la posterior definición de objetos de la misma.

Desarrollaremos una clase que represente un cliente de un banco.
La clase cliente tiene como atributos:

```
nombre
saldo
```

y las responsabilidades o métodos de la clase son:

```
Constructor (inicializamos los atributos del objeto)
depositar
extraer
```

Para acostumbrarnos a trabajar en un archivo separado, implementamos 'pagina2.js' que contiene la clase cliente con sus métodos (funciones) y sus atributos (variables):

```
function cliente(nombre,saldo)
{
  this.nombre=nombre;
  this.saldo=saldo;
  this.depositar=depositar;
  this.extraer=extraer;
}

function depositar(dinero)
{
  this.saldo=this.saldo+dinero;
}

function extraer(dinero)
{
  this.saldo=this.saldo-dinero;
}
```

El nombre de la clase coincide con el nombre de la función principal que implementamos (también llamado constructor de la clase):

```
function cliente(nombre,saldo)
{
  this.nombre=nombre;
  this.saldo=saldo;
  this.depositar=depositar;
  this.extraer=extraer;
}
```

A esta función llegan como parámetro los valores con que queremos inicializar los atributos. Con la palabra clave 'this' diferenciamos los atributos de los parámetros (los atributos deben llevar la palabra clave this)

```
  this.nombre=nombre;
  this.saldo=saldo;
```

También en el constructor inicializamos la referencia a todos los métodos que contendrá la clase:

```
  this.depositar=depositar;
  this.extraer=extraer;
```

Por último, implementamos todos los métodos de la clase:

```
function depositar(dinero)
{
  this.saldo=this.saldo+dinero;
}

function extraer(dinero)
{
  this.saldo=this.saldo-dinero;
}
```

De nuevo recordemos que diferenciamos los atributos de la clase por la palabra clave this.

Ahora veamos el archivo HTML que incorpora el archivo JS y define un objeto de la clase planteada:

```
<html>
<head>
<title>Problema</title>
<script languaje="javascript" src="pagina2.js" type="text/javascript">
</script>
</head>
<body>
<script languaje="javascript">
var cliente1;
cliente1=new cliente('diego',1200);
document.write('Nombre del cliente:'+cliente1.nombre+'<br>');
document.write('Saldo actual:'+cliente1.saldo+'<br>');
cliente1.depositar(120);
document.write('Saldo luego de depositar $120----
>'+cliente1.saldo+'<br>');
cliente1.extraer(1000);
document.write('Saldo luego de extraer $1000----
>'+cliente1.saldo+'<br>');
</script>
</body>
</html>
```

Recordemos que lo primero que hacemos, según lo visto en conceptos anteriores, es importar el archivo *.js:

```
<script languaje="javascript" src="pagina2.js" type="text/javascript">
```

Luego, la sintaxis para la creación de un objeto de la clase cliente es:

```
var cliente1;
cliente1=new cliente('diego',1200);
```

Similar a conceptos anteriores cuando definiamos objetos de la clase Date o Array. Con el operador new se crea un objeto de la clase cliente y se llama inmediatamente el constructor de la clase. El constructor retorna una referencia del objeto que se almacena en la variable cliente1.
De ahí en más podemos acceder a los atributos y llamar a los métodos del objeto cliente1 de la clase cliente:

```
document.write('Nombre del cliente:'+cliente1.nombre+'<br>');
document.write('Saldo actual:'+cliente1.saldo+'<br>');
cliente1.depositar(120);
document.write('Saldo luego de depositar $120----
>'+cliente1.saldo+'<br>');
cliente1.extraer(1000);
document.write('Saldo luego de extraer $1000----
>'+cliente1.saldo+'<br>');
```

Podemos decir que la ventaja que podemos obtener con el planteo de clases es hacer nuestros programas mucho más organizados, entendibles y fundamentalmente, poder reutilizar clases en distintos proyectos