**Programación orientada a objetos JavaScript.**

Un objeto es una estructura que contiene tanto las variables (llamadas propiedades) como las funciones que manipulan dichas variables (llamadas métodos). A partir de esta estructura se ha creado un nuevo modelo de programación (la programación orientada a objetos) que atribuye a los mismos propiedades como herencia o polimorfismo. Como veremos, JavaScript simplifica en algo este modelo y hace una programación híbrida entre la programación estructurada y la programación orientada a objetos.

El modelo de la programación orientada a objetos normal y corriente separa los mismos en dos: clases e instancias (objetos). Las primeras son entes más abstractos que definen un conjunto determinado de objetos. Las segundas son miembros de una clase, poseyendo las mismas propiedades que la clase a la cual pertenecen.

**Propiedades y métodos.**

Para acceder a los métodos y propiedades de un objeto debemos utilizar la siguiente sintaxis:

objeto.propiedad

objeto.metodo(parametros)

**Conceptos Básicos.**

**Objetos**

Son todas las cosas con identidad propia. Se relacionan entre si. Poseen características (atributos) y tienen responsabilidades (funciones, métodos) que deben cumplir. Son ejemplares (instancias) de una clase y conocen a la clase a la cual pertenecen.

**Atributos o propiedades**

Son las características, cualidades distintivas de cada objeto. Deben ser mínimos para poder realizar todas las operaciones que requiere la aplicación.

Ejemplos de objetos del mundo real:

- Casa:

atributos: tamaño, precio, cantidad de habitaciones, etc.;

responsabilidades: comodidad, seguridad, etc.

- Mesa:

atributos: altura, largo, ancho, etc.;

responsabilidades: contener elementos.

- Ventana:

atributos: tamaño, color, etc.;

responsabilidades: abrirse, cerrarse, etc.

Ejemplos de objetos del mundo de la programación:

- Ventana:

atributos: tamaño, color, etc.;

responsabilidades: mostrar título,achicarse

etc.

**Responsabilidades o Métodos.**

Son las responsabilidades que debe cumplir la clase. El objetivo de un método es ejecutar las actividades que tiene encomendada la clase.

Es un algoritmo (conjunto de operaciones) que se ejecuta en respuesta a un mensaje; respuestas a mensajes para satisfacer peticiones.

Un método consiste en el nombre de la operación y sus argumentos. El nombre del método identifica una operación que se ejecuta.

Un método está determinado por la clase del objeto receptor, todos los objetos de una clase usan el mismo método en respuesta a mensajes similares.

La interpretación de un mensaje (selección del método ejecutado) depende del receptor y puede variar con distintos receptores, es decir, puede variar de una clase a otra.

**Clases**

Una clase es un molde para objetos que poseen las mismas características (que pueden recibir los mismos mensajes y responden de la misma manera).

Una clase es una representación de una idea o concepto. Unidad que encapsula códigos y datos para los métodos (operaciones).

Todos los ejemplares de una clase se comportan de forma similar (invocan el mismo método) en respuesta a mensajes similares.

La clase a la cual pertenece un objeto determina el comportamiento del objeto.

Una clase tiene encomendadas actividades que ejecutan los métodos.

Las clases están definidas por:

- Atributos (Propiedades),

- Comportamiento (operaciones o métodos) y

- Relaciones con otros objetos.

Una aplicación es un conjunto de objetos de determinadas clases.

**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:

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 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>

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 type="text/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 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.

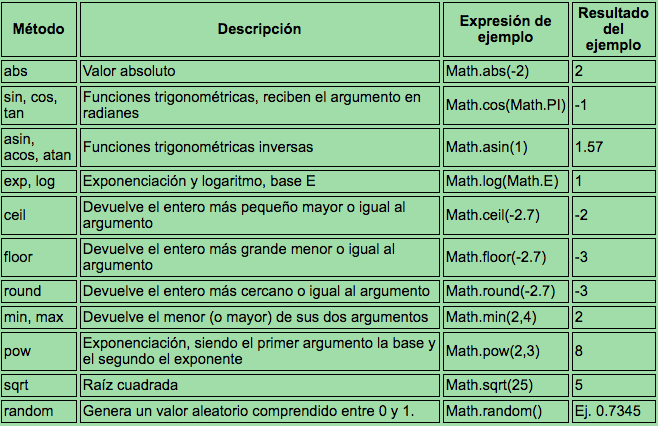
Es importante notar que cuando imprimimos, disponemos como subíndice el valor ingresado menos 1, esto debido a que normalmente el operador de nuestro programa carga un valor comprendido entre 1 y 12. Recordar que los vectores comienzan a numerarse a partir de la componente cero.

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



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)

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 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();

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 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 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>

</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:

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.

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 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. 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 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 si)

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.

**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.

**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 propiedad 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.

**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>

</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 pinta 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>

**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();

}

**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></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()

{

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>

**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 entre al periódico La Nación, pero primero muestre una ventana de confirmación, si queremos ingresar al periódico. En caso que el visitante presione el botón afirmativo, redireccionamos al sitio de La Nación, en caso contrario mostramos un mensaje:

<html>

<head>

</head>

<body>

<script type="text/javascript">

function verificarEntrada()

{

if (window.confirm('Desea salir del sitio e ingresar al periódico La Nación?'))

{

window.location='http://www.lanacion.com.ar';

}

else

{

window.alert('No hay problema');

}

}

</script>

<a href="javascript:verificarEntrada()">Ingresar a La Nación</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:verificarEntrada()">Ingresar a La Nación</a>

La función verificarEntrada() 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('Desea salir del sitio e ingresar al periódico La Nación?'))

{

window.location='http://www.lanacion.com.ar';

}

else

{

window.alert('No hay problema');

}

**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 type="text/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 type="text/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 type="text/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 type="text/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>