Indice

Eventos:

En JavaScript, la interacción con el usuario se consigue mediante la captura de los eventos que éste produce. Un evento es una acción del usuario ante la cual puede realizarse algún proceso (por ejemplo, el cambio del valor de un formulario, o la pulsación de un enlace).

Los eventos se capturan mediante los manejadores de eventos. El proceso a realizar se programa mediante funciones JavaScript llamadas por los manejadores de eventos.
La siguiente tabla muestra los manejadores de eventos que pueden utilizarse en JavaScript, la versión a partir de la cual están soportados y su significado.

Manejador	Versión	Se produce cuando...
onAbort	1.1	El usuario interrumpe la carga de una imagen
onBlur	1.0	Un elemento de formulario, una ventana o un marco pierden el foco
onChange	1.0 (1.1 para *FileUpload*)	El valor de un campo de formulario cambia
onClick	1.0	Se hace *click* en un objeto o formulario
onDblClick	1.2 (no en Mac)	Se hace *click* doble en un objeto o

		formulario
onDragDrop	1.2	El usuario arrastra y suelta un objeto en la ventana
onError	1.1	La carga de un documento o imagen produce un error
onFocus	1.1 (1.2 para *Layer*)	Una ventana, marco o elemento de formulario recibe el foco
onKeyDown	1.2	El usuario pulsa una tecla
onKeyPress	1.2	El usuario mantiene pulsada una tecla
onKeyUp	1.2	El usuario libera una tecla
onLoad	1.0 (1.1 para *image*)	El navegador termina la carga de una ventana
onMouseDown	1.2	El usuario pulsa un botón del ratón
onMouseMove	1.2	El usuario mueve el puntero
onMouseOut	1.1	El puntero abando una área o enlace
onMouseOver	1.0 (1.1 para *area*)	El puntero entra en una área o imagen
onMouseUp	1.2	El usuario libera un botón del ratón
onMove	1.2	Se mueve una ventana o un marco
onReset	1.1	El usuario limpia un formulario
onResize	1.2	Se cambia el tamaño de una ventana o marco
onSelect	1.0	Se selecciona el texto del campo texto o área de texto de un formulario
onSubmit	1.0	El usuario envía un formulario
onUnload	1.0	El usuario abandona una página

Normalmente, los manejadores de eventos requieren información adicional para procesar sus tareas. Si una función por ejemplo se encarga de procesar el evento onclick, quizás necesite saber en que posición estaba el ratón en el momento de pinchar el botón.

No obstante, el caso más habitual en el que es necesario conocer información adicional sobre el evento es el de los eventos asociados al teclado. Normalmente, es muy importante conocer la tecla que se ha pulsado, por ejemplo para diferenciar las teclas normales de las teclas especiales (ENTER, tabulador, Alt, Ctrl., etc.).
JavaScript permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado event. Desafortunadamente, los diferentes navegadores presentan diferencias muy notables en el tratamiento de la información sobre los eventos.
La principal diferencia reside en la forma en la que se obtiene el objeto event. Internet Explorer considera que este objeto forma parte del objeto window y el resto de navegadores lo consideran como el único argumento que tienen las funciones manejadoras de eventos.
Aunque es un comportamiento que resulta muy extraño al principio, todos los navegadores modernos excepto Internet Explorer crean *mágicamente* y de forma automática un argumento que se pasa a la función manejadora, por lo que no es necesario incluirlo en la llamada a la función manejadora. De esta forma, para utilizar este *"argumento mágico"*, sólo es necesario asignarle un nombre, ya que los navegadores lo crean automáticamente.
En resumen, en los navegadores tipo Internet Explorer, el objeto event se obtiene directamente mediante:

```
var evento = window.event;
```

Por otra parte, en el resto de navegadores, el objeto event se obtiene *mágicamente* a partir del argumento que el navegador crea automáticamente:

```
function manejadorEventos(elEvento) {
 var evento = elEvento;
}
```

Si se quiere programar una aplicación que funcione correctamente en todos los navegadores, es necesario obtener el objeto event de forma correcta según cada navegador. El siguiente código muestra la forma correcta de obtener el objeto event en cualquier navegador:

```
function manejadorEventos(elEvento) {
 var evento = elEvento || window.event;
}
```

Una vez obtenido el objeto event, ya se puede acceder a toda la información relacionada con el evento, que depende del tipo de evento producido.

Información sobre el evento

La propiedad type indica el tipo de evento producido, lo que es útil cuando una misma función se utiliza para manejar varios eventos:

```
var tipo = evento.type;
```

La propiedad type devuelve el tipo de evento producido, que es igual al nombre del evento pero sin el prefijo on.
Mediante esta propiedad, se puede rehacer de forma más sencilla el ejemplo anterior en el que se resaltaba una sección de contenidos al pasar el ratón por encima:

```
function resalta(elEvento) {
 var evento = elEvento || window.event;
 switch(evento.type) {
   case 'mouseover':
```

```
    this.style.borderColor = 'black';
    break;
  case 'mouseout':
    this.style.borderColor = 'silver';
    break;
 }
}

window.onload = function() {
 document.getElementById("seccion").onmouseover = resalta;
 document.getElementById("seccion").onmouseout = resalta;
}

<div id="seccion" style="width:150px; height:60px; border:thin solid silver">
 Sección de contenidos...
</div>
```

mas acerca del evento:
http://www.quirksmode.org/js/events_properties.html
attachar eventos:
http://www.anieto2k.com/2006/10/16/gestion-de-eventos-en-javascript/

Array:Métodos

concat()
se emplea para concatenar los elementos de varios arrays

```
var array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6);   // array2 = [1, 2, 3, 4, 5, 6]
array3 = array1.concat([4, 5, 6]); // array3 = [1, 2, 3, 4, 5, 6]
```

join(separador),
Es la función contraria a split(). Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado

```
var array = ["hola", "mundo"];
var mensaje = array.join(""); // mensaje = "holamundo"
mensaje = array.join(" ");    // mensaje = "hola mundo"
```

pop(),
Elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
var array = [1, 2, 3];
```

```
var ultimo = array.pop();
// ahora array = [1, 2], ultimo = 3
```

push(),
añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.push(4);
// ahora array = [1, 2, 3, 4]
```

shift(),
elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento.

```
var array = [1, 2, 3];
var primero = array.shift();
// ahora array = [2, 3], primero = 1
```

unshift(),
añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.unshift(0);
// ahora array = [0, 1, 2, 3]
```

reverse(),
modifica un array colocando sus elementos en el orden inverso a su posición original:

```
var array = [1, 2, 3];
array.reverse();
// ahora array = [3, 2, 1]
```

FOR.

El bucle FOR se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número

de veces que queremos que se ejecute. La sintaxis del bucle for se muestra a continuación.

```
for (inicialización; condición; actualización) {
        //sentencias a ejecutar en cada iteración
}
```

El bucle FOR tiene tres partes incluidas entre los paréntesis, que nos sirven para definir cómo deseamos que se realicen las repeticiones. La primera parte es la inicialización, que se ejecuta solamente al comenzar la primera iteración del bucle. En esta parte se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle.

La segunda parte es la condición, que se evaluará cada vez que comience una iteración del bucle. Contiene una expresión para decidir cuándo se ha de detener el bucle, o mejor dicho, la condición que se debe cumplir para que continúe la ejecución del bucle.

Por último tenemos la actualización, que sirve para indicar los cambios que queramos ejecutar en las variables cada vez que termina la iteración del bucle, antes de comprobar si se debe seguir ejecutando.

Después del for se colocan las sentencias que queremos que se ejecuten en cada iteración, acotadas entre llaves.

Un ejemplo de utilización de este bucle lo podemos ver a continuación, donde se imprimirán los números del 0 al 10.

```
var i
for (i=0;i<=10;i++) {
        document.write(i)
        document.write("<br>")
}
```

En este caso se inicializa la variable i a 0. Como condición para realizar una iteración, se tiene que cumplir que la variable i sea menor o igual que 10. Como actualización se incrementará en 1 la variable i.

Como se puede comprobar, **este bucle es muy potente, ya que en una sola línea podemos indicar muchas cosas distintas y muy variadas**, lo que permite una rápida configuración del bucle y una versatilidad enorme.

Por ejemplo si queremos escribir los número del 1 al 1.000 de dos en dos se escribirá el siguiente bucle.

```
for (i=1;i<=1000;i+=2)
        document.write(i)
```

Si nos fijamos, en cada iteración actualizamos el valor de i incrementándolo en 2 unidades.
Nota: Otro detalle, no utilizamos las llaves englobando las instrucciones del bucle FOR porque sólo tiene una sentencia y en este caso no es obligado, tal como pasaba con las instrucciones del IF.
Si queremos contar descendentemente del 343 al 10 utilizaríamos este bucle.

```
for (i=343;i>=10;i--)
        document.write(i)
```

En este caso decrementamos en una unidad la variable i en cada iteración, comenzando en el valor 343 y siempre que la variable tenga un valor mayor o igual que 10.

Ejercicio de ejemplo del bucle for

Vamos a hacer una pausa para asimilar el bucle for con un ejercicio que no encierra ninguna dificultad si hemos entendido el funcionamiento del bucle.

Se trata de hacer un bucle que escriba en una página web los encabezamientos desde <H1> hasta <H6> con un texto que ponga "Encabezado de nivel x".

Lo que deseamos escribir en una página web mediante Javascript es lo siguiente:

```
<H1>Encabezado de nivel 1</H1>
<H2>Encabezado de nivel 2</H2>
<H3>Encabezado de nivel 3</H3>
<H4>Encabezado de nivel 4</H4>
<H5>Encabezado de nivel 5</H5>
<H6>Encabezado de nivel 6</H6>
```

Para ello tenemos que hacer un bucle que empiece en 1 y termine en 6 y en cada iteración escribiremos el encabezado que toca.

```
for (i=1;i<=6;i++) {
        document.write("<H" + i + ">Encabezado de nivel " + i + "</H" + i + ">")
}
```

Programacion Orientada a Objetos

No todos los desarrolladores están de acuerdo sobre si JavaScript es un lenguaje de programación orientado a objetos o no, pero es indiscutible que es una herramienta excelente para el desarrollo de aplicaciones que trabajen con trabajar con clases y objetos.

Clases y Objetos

Para entender la programacion orientada a objetos debemos primero entender lo que es una clase, pero primero diremos que un objeto es un conjunto de atributos y metodos agrupados.

Una clase es un grupo de objetos que comparten los mismos atributos y métodos, veamos como podemos crear una clase llamada *popup*, como observareis no hay ninguna diferéncia con una funcion solo que al crearlo usamos **new**:

```
<script language="javascript">
// Creamos la clase
function popup ( ) {
 // ...
}
ventana = new popup ();
</script>
```

Los atributos

La clase que acabamos de crear tendrá sus atributos, que podrán ser publicos (accesibles desde fuera del objeto) o privados (solo accesibles desde el código del objeto).

Vamos a añadir un atributo público a la clase que hemos creado para indicar la URL del popup:

```
<script language="javascript">
// Creamos la clase
function popup ( ) {
 // Atributo público inicializado a about:blank
 this.url = 'about:blank';
 // ...
}
ventana = new popup ();
</script>
```

Y le añadimos también una clase privada para almacenar el objeto window de la ventana abierta.

```
<script language="javascript">
// Creamos la clase
function popup ( ) {
 // Atributo público inicializado a about:blank
 this.url = 'about:blank';
 // Atributo privado para el objeto window
```

```
 var ventana = null;
 // ...
}
ventana = new popup ();
ventana.url = 'http://www.programacionweb.net/';
</script>
```

Los métodos

Al crear un objeto de tipo imagen se ejecutará el metodo **constructor** de la imagen que será en el caso de nuestro ejemplo la funcion *popup*, cuando se ejecuta, este crea el objeto y todas sus variables (atributos) y funciones (metodos).

En el interior de esta función constructor, podemos crear nuevas funciones públicas y privadas que a su vez, podrían ser constructor de un objeto 'hijo', vamos a completar el ejemplo anterior añadiendo un par de metodos.

```
<script language="javascript">
function popup ( ) {
 // Atributo público inicializado a about:blank
 this.url = 'about:blank';
 // Atributo privado para el objeto window
 var ventana = null;
 // Metodo público para abrir el popup
 this.abrir = function ( ) {
 // Generamos la ventana
 ventana = window.open ( this.url );
 // Si no hay ventana llamamos al error
 if ( ! ventana ) error ( 'El popup ha sido bloqueado' );
 }
 // Metodo privado para alertar un mensaje en caso de error
 var error = function ( texto ) {
 // Mostramos el error
 alert ( texto );
 }
}
```

```
ventana = new popup ();
ventana.url = 'http://www.programacionweb.net/';
ventana.abrir ();
</script>
```

La visibilidad de los elementos

Los elementos de un objeto (metodos y clases) són accesibles desde el exterior del mismo objeto cuando los creamos con **this.** tal y como hemos observado en el ejemplo con *url* y *abrir*:

```
<script language="JavaScript">
// ...
ventana.url = 'http://www.programacionweb.net/';
ventana.abrir ();
</script>
```

Por otro lado tambien hemos creado los objetos *ventana* y *error*, pero al crearlos con **var** solo serán accesibles desde el interior de la clase y sus subclases y funciones:

```
<script language="JavaScript">
// ...
ventana.url = 'http://www.programacionweb.net/';
// Llamamos a un metodo privado
// Error: ventana.error is not a function
ventana.error ( 'Hola amigo' );
</script>
```