3.- Eventos.

Caso práctico
La mayor parte de las veces que un usuario realiza acciones en un formulario está generando eventos. Por ejemplo, cuando hace click con el ratón, cuando sitúa el cursor en un campo, cuando mueve el ratón sobre algún objeto, etc.
Con JavaScript podremos programar que, cuando se produzca alguno de esos eventos, realice una tarea determinada. Es lo que se conoce en programación como ***capturar un evento****.*
Los modelos de registro de esos eventos, así como el orden en el que esos eventos se generan, es la parte que va a estudiar ***Antonio*** *ahora mismo. Esta parte le ayudará mucho cuando tenga que capturar eventos que se produzcan en secuencia, o cuando quiera cancelar un evento, etc.*

Hay que tener en cuenta que, sin eventos prácticamente no hay scripts. En casi todas las páginas web que incorporan JavaScript, suele haber eventos programados que disparan la ejecución de dichos scripts. La razón es muy simple, JavaScript fue diseñado para añadir interactividad a las páginas: el usuario realiza algo y la página reacciona.

Por lo tanto, JavaScript necesita detectar de alguna forma las acciones del usuario para saber cuándo reaccionar. También necesita saber las funciones, que queremos que ejecute cuando se produzcan esas acciones.

Cuando el usuario hace algo se produce un evento. También habrá algunos eventos que no están relacionados directamente con acciones de usuario: por ejemplo el evento de carga (`load`) de un documento, que se producirá automáticamente cuando un documento ha sido cargado.

Todo el tema de gestión de eventos se popularizó a raíz de la versión 2 de Netscape, que también soportaba eventos. Netscape 2 soportaba solamente algunos eventos. Los eventos `mouseOver` y `mouseOut`, se hicieron muy famosos a raíz de su utilización para hacer el efecto de sustitución de una imagen por otra al pasar el ratón por encima. Todo el resto de navegadores, incluido Explorer, tuvieron que adaptarse a la forma en la que Netscape 2 y 3 gestionaban los eventos.

Aunque hoy en día la técnica de gestión de eventos varía con el objetivo de independizar el código de JavaScript de la estructura HTML, los navegadores todavía son compatibles con las técnicas utilizadas por Netscape.

Anteriormente, a las versiones 4 de los navegadores, los eventos (interacciones del usuario y del sistema), eran capturados preferentemente por gestores de eventos definidos como atributos en las etiquetas HTML (modelo de eventos en línea). Por ejemplo, cuando un usuario hacía click en un botón, el evento de `onclick` que se había programado en la etiqueta HTML era disparado. Ese evento hacía una llamada a una función en la que se realizarían las operaciones programadas por el usuario.

Aunque todo ese modo de gestión de eventos sigue funcionando, los navegadores modernos ya incorporan un modelo de eventos, que proporciona un montón de información sobre cómo ocurre un evento. Estas propiedades son accesibles a través de JavaScript, permitiendo programar respuestas más inteligentes a las interacciones del usuario con los objetos del documento.

Incompatibilidades entre navegadores

Muchas veces, lo que se hacía en un principio antes de programar cualquier evento, era detectar qué navegador estábamos utilizando, para saber si nuestro navegador soportaba, o no, los métodos y propiedades que queríamos usar. Por ejemplo:

```
if (Netscape) {
   utilizar modelo Netscape
}
else if (Explorer) {
```

```
   utilizar modelo Microsoft
}
```

Pero hoy en día ni ésto ni el modelo de detección basado en DHTML se recomiendan debido a las diferencias que hay entre todos los navegadores actuales. Por lo tanto hay que intentar usar modelos de detección de eventos estándar y que sean los navegadores los que tengan que adaptarse a ese modelo.

3.1.- Modelo de registro de eventos en línea.

En el modelo de registro de eventos en línea (estandarizado por Netscape), el evento es añadido como un atributo más a la etiqueta HTML, como por ejemplo:

```
<A href="pagina.html" onClick="alert('Has pulsado en el enlace')">Pulsa aqui</a>
```

Cuando hacemos click en el enlace, se llama al gestor de eventos `onClick` (al hacer click) y se ejecuta el script; que contiene en este caso una alerta de JavaScript. También se podría realizar lo mismo pero llamando a una función:

```
<A href="pagina.html" onClick="alertar()">Pulsa aqui</a>
function alertar(){
     alert("Has pulsado en el enlace");
}
```

La mezcla de minúsculas y mayúsculas en los nombres de evento (`onClick`, `onMouseOver`) es sólo por tradición, ya que HTML es insensible a mayúsculas y minúsculas. En cambio en XHTML, sí que los atributos tendrán que ir obligatoriamente siempre en minúsculas: tienes que escribir `onclick` y `onmouseover`. Es muy importante que te fijes en esto, ya que probablemente trabajarás con XHTML y deberás cumplir el estándar: "**todos los nombres de atributos irán siempre en minúscula**".

NO USES EL MODELO DE REGISTRO DE EVENTOS EN LÍNEA

Este modelo no se recomienda, y aunque lo has visto en ejemplos que hemos utilizado hasta ahora, tiene el problema de que estamos mezclando la estructura de la página web con la programación de la misma, y lo que se intenta hoy en día es separar la programación en JavaScript, de la estructura HTML, por lo que este modelo no nos sirve.

En el ejemplo anterior, cuando haces click en el enlace se mostrará la alerta y a continuación te conectará con la pagina.html. En ese momento desaparecerán de memoria los objetos que estaban en un principio, cuando se programó el evento. Esto puede ser un problema, ya que si por ejemplo la función a la que llamamos, cuando se produce el evento, tiene que realizar varias tareas, éstas tendrían que ser hechas antes de que nos conecte con la nueva página.

Éste modo de funcionamiento ha sido un principio muy importante en la gestión de eventos. Si un evento genera la ejecución de un script y además también se genera la acción por defecto para ese objeto entonces:

1. El script se ejecutará primero.
2. La acción por defecto se ejecutará después.

EVITAR LA ACCIÓN POR DEFECTO

A veces es interesante el bloquear o evitar que se ejecute la acción por defecto. Por ejemplo, en nuestro caso anterior podríamos evitar que nos conecte con la nueva pagina.html. Cuando programamos un gestor de eventos, ese gestor podrá devolver un valor booleano `true` o `false`. Eso tendremos que programarlo con la instrucción `return true|false`. `False` quiere decir "no ejecutes la acción por defecto". Por lo tanto nuestro ejemplo quedará del siguiente modo:

```
<A href="pagina.html" onClick="alertar(); return false">Pulsa aqui</a>
```

De esa forma, cada vez que pulsemos en el enlace realizará la llamada a la función `alertar()` y cuando termine ejecutará la instrucción "`return false`", que le indicará al navegador que no ejecute la acción por defecto asignada a ese objeto (en este caso la acción por defecto de un hiperenlace es conectarnos con la página `href` de destino).

También sería posible que nos preguntara si queremos o no queremos ir a la pagina.html. Eso podríamos hacerlo sustituyendo `true` o `false` por una función que devuelva `true` o `false` según la respuesta que demos al "`confirm`":

```
<A href="pagina.html" onClick="return preguntar()">Pulsa aqui</a>

function preguntar(){
     return confirm("¿Desea realmente ir a esa dirección?");
}
```

3.2.- Modelo de registro de eventos tradicional.

En los navegadores antiguos, el modelo que se utilizaba era el modelo en línea. Con la llegada de DHTML, el modelo se extendió para ser más flexible. En este nuevo modelo el evento pasa a ser una propiedad del elemento, así que por ejemplo los navegadores modernos ya aceptan el siguiente código de JavaScript:

```
elemento.onclick = hacerAlgo;  // cuando el usuario haga click en el objeto, se llamará a la función hacerAlgo()
```

Esta forma de registro, no fue estandarizada por el W3C, pero debido a que fue ampliamente utilizada por Netscape y Microsoft, todavía es válida hoy en día. La ventaja de este modelo es que podremos asignar un evento a un objeto desde JavaScript, con lo que ya estamos separando el código de la estructura. Fíjate que aquí los nombres de los eventos sí que van siempre en minúsculas.

```
elemento.onclick = hacerAlgo;
```

Para eliminar un gestor de eventos de un elemento u objeto, le asignaremos `null`:

```
elemento.onclick = null;
```

Otra gran ventaja es que, como el gestor de eventos es una función, podremos realizar una llamada directa a ese gestor, con lo que estamos disparando el evento de forma manual. Por ejemplo:

```
elemento.onclick();
// Al hacer ésto estamos disparando el evento click de forma manual y se ejecutará la función hacerAlgo()
```

SIN PARÉNTESIS

Fíjate que en el registro del evento no usamos paréntesis (). El método `onclick` espera que se le asigne una función completa. Si haces: `element.onclick = hacerAlgo()`; la función será ejecutada y el resultado que devuelve esa función será asignado a `onclick`. Pero ésto no es lo que queremos que haga, queremos que se ejecute la función cuando se dispare el evento.

USO DE LA PALABRA RESERVADA THIS

En el modelo en línea podemos utilizar la palabra reservada `this` cuando programamos el gestor de eventos. Por ejemplo:

```
<A href="pagina.html" ID="mienlace" onClick="alertar(this)">Pulsa aqui</a>
<script type="text/javascript">
   function alertar(objeto){
      alert("Te conectaremos con la página: "+objeto.href);
   }
```

```
</script>

Su equivalente en el <strong>modelo tradicional</strong> sería:
<A id="mienlace" href="pagina.html">Pulsa aqui</a>

<script type="text/javascript">
   document.getElementById("mienlace").onclick = alertar;

   function alertar(){
      alert("Te conectaremos con la página: "+this.href);
   }
</script>
```

Fíjate que estamos usando `this` dentro de la función, sin pasar ningún objeto (tal y como hacíamos en el modelo en línea). En el modelo tradicional el `this` que está dentro de la función, hace referencia al objeto donde hemos programado el evento. También hay que destacar que en este modelo es importante que el hiperenlace sea declarado antes de programar la asignación `onclick`, ya que si por ejemplo escribimos el hiperenlace después del bloque de código de JavaScript, éste no conocerá todavía el objeto y no le podrá asignar el evento de `onclick`. Esto también se podría solucionar, programando la asignación de eventos a los objetos, en una función que se ejecute cuando el documento esté completamente cargado. Por ejemplo con `window.onload=asignarEventos`.

Si por ejemplo queremos que cuando se produzca el evento se realicen llamadas a más de una función lo podremos hacer de la siguiente forma:

```
elemento.onclick = function {llamada1; llamada2 };
```

3.3.- Modelo de registro avanzado de eventos según W3C.

El W3C en la especificación del DOM de nivel 2, pone especial atención en los problemas del modelo tradicional de registro de eventos. En este caso ofrece una manera sencilla de registrar los eventos que queramos, sobre un objeto determinado.

La clave para poder hacer todo eso está en el método `addEventListener()`.

Este método tiene tres **argumentos**: el **tipo de evento**, la **función a ejecutar** y un **valor** booleano (`true` o `false`), que se utiliza para indicar cuándo se debe capturar el evento: en la fase de captura (`true`) o de burbujeo (`false`). En el apartado 3.5 veremos en detalle la diferencia entre estas dos fases.

```
elemento.addEventListener('evento', función, false|true)
```

Por ejemplo para registrar la función `alertar()` de los ejemplos anteriores, haríamos:

```
document.getElementById("mienlace").addEventListener('click',alertar,false);

function alertar(){
     alert("Te conectaremos con la página: "+this.href);
}
```

La ventaja de este método, es que podemos añadir tantos eventos como queramos. Por ejemplo:

```
document.getElementById("mienlace").addEventListener('click',alertar,false);
document.getElementById("mienlace").addEventListener('click',avisar,false);
document.getElementById("mienlace").addEventListener('click',chequear,false);
```

Por lo tanto, cuando hagamos click en "mienlace" se disparará la llamada a las tres funciones. Por cierto, el W3C no indica el orden de disparo, por lo que no sabemos cuál de las tres funciones se ejecutará primero. **Fíjate también, que el nombre de los eventos al usar `addEventListener` no lleva `'on'` al comienzo**.

También se pueden usar funciones anónimas (sin nombre de función), con el modelo W3C:

```
element.addEventListener('click', function () {
     this.style.backgroundColor = '#cc0000';
```

```
}, false)
```

Uso de la palabra reservada this

La palabra reservada `this`, tiene exactamente la misma funcionalidad que hemos visto en el modelo tradicional.

¿QUÉ EVENTOS HAN SIDO REGISTRADOS?

Uno de los problemas de la implementación del modelo de registro del W3C, es que no podemos saber con antelación, los eventos que hemos registrado a un elemento.

En el modelo tradicional si hacemos: `alert(elemento.onclick)`, nos devuelve `undefined`, si no hay funciones registradas para ese evento, o bien el nombre de la función que hemos registrado para ese evento. Pero en este modelo no podemos hacer eso.

El **W3C** en el reciente **nivel 3 del DOM**, introdujo un método llamado `eventListenerList`, que almacena una lista de las funciones que han sido registradas a un elemento. Tienes que tener cuidado con este método, porque no está soportado por todos los navegadores.

Para eliminar un evento de un elemento, usaremos el método `removeEventListener()`:

```
elemento.removeEventListener('evento', función, false|true);
```

Para cancelar un evento, este modelo nos proporciona el método `preventDefault()`.

3.4.- Modelo de registro de eventos según Microsoft.

Microsoft también ha desarrollado un modelo de registro de eventos. Es similar al utilizado por el W3C, pero tiene algunas modificaciones importantes.

Para **registrar un evento**, tenemos que enlazarlo con `attachEvent()`:

```
elemento.attachEvent('onclick', hacerAlgo);
```

o si necesitas dos gestores del mismo evento:

```
elemento.attachEvent('onclick', hacerUnaCosa);
elemento.attachEvent('onclick', hacerOtraCosa);
```

Para **eliminar un evento**, se hará con `detachEvent()`:

```
elemento.detachEvent('onclick', hacerAlgo);
```

Comparando este modelo con el del W3C tenemos dos diferencias importantes:

- ✓ Los eventos siempre burbujean, no hay forma de captura.
- ✓ La función que gestiona el evento está referenciada, no copiada, con lo que la palabra reservada `this` siempre hace referencia a window y será completamente inútil.

Como resultado de estas dos debilidades, cuando un evento burbujea hacia arriba es imposible conocer cuál es el elemento HTML que gestionó ese evento.

Listado de atributos de eventos (**IE**: Internet Explorer, **F**: Firefox, **O**: Opera, **W3C**: W3C Standard.)

Listado de atributos de eventos					
Atributo	**El evento se produce cuando...**	**IE**	**F**	**O**	**W3C**
onblur	Un elemento pierde el foco.	3	1	9	**Sí**
onchange	El contenido de un campo cambia.	3	1	9	**Sí**
onclick	Se hace click con el ratón en un objeto.	3	1	9	**Sí**
ondblclick	Se hace doble click con el ratón sobre un objeto.	4	1	9	**Sí**

onerror	Ocurre algún error cargando un documento o una imagen.	4	1	9	**Sí**
onfocus	Un elemento tiene el foco.	3	1	9	**Sí**
onkeydown	Se presiona una tecla del teclado.	3	1	No	**Sí**
onkeypress	Se presiona una tecla o se mantiene presionada.	3	1	9	**Sí**
onkeyup	Cuando soltamos una tecla.	3	1	9	**Sí**
onload	Una página o imagen terminaron de cargarse.	3	1	9	**Sí**
onmousedown	Se presiona un botón del ratón.	4	1	9	**Sí**
onmousemove	Se mueve el ratón.	3	1	9	**Sí**
onmouseout	Movemos el ratón fuera de un elemento.	4	1	9	**Sí**
onmouseover	El ratón se mueve sobre un elemento.	3	1	9	**Sí**
onmouseup	Se libera un botón del ratón.	4	1	9	**Sí**
onresize	Se redimensiona una ventana o frame.	4	1	9	**Sí**
onselect	Se selecciona un texto.	3	1	9	**Sí**
onunload	El usuario abandona una página.	3	1	9	**Sí**

Más información sobre eventos de teclado, ratón y otros eventos
http://www.w3schools.com/jsref/dom_obj_event.asp

3.5.- Orden de disparo de los eventos.

Imagina que tenemos un elemento contenido dentro de otro elemento, y que tenemos programado el mismo tipo de evento para los dos (por ejemplo el evento click). ¿Cuál de ellos se disparará primero? Sorprendentemente, esto va a depender del tipo de navegador que tengamos.

El problema es muy simple. Imagina que tenemos el siguiente gráfico:

y ambos tienen programado el evento de click. Si el usuario hace click en el elemento2, provocará un click en ambos: elemento1 y elemento2. ¿Pero cuál de ellos se disparará primero?, ¿cuál es el orden de los eventos?

Tenemos **dos Modelos propuestos** por Netscape y Microsoft en sus orígenes:

- ✓ Netscape dijo que, el evento en el elemento1 tendrá lugar primero. Es lo que se conoce como **"captura de eventos**".
- ✓ Microsoft mantuvo que, el evento en el elemento2 tendrá precedencia. Es lo que se conoce como "**burbujeo de eventos**".

Los dos modelos son claramente opuestos. Internet Explorer sólo soporta burbujeo (bubbling). Mozilla, Opera 7 y Konqueror soportan los dos modelos. Y las versiones antiguas de Opera e iCab no soportan ninguno.

Modelo W3C

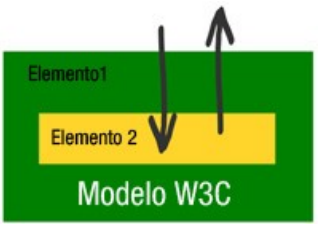
W3C decidió tomar una posición intermedia. Así supone que, cuando se produce un evento en su modelo de eventos, primero se producirá la fase de captura hasta llegar al elemento de destino, y luego se producirá la fase de burbujeo hacia arriba. Este modelo es el estándar, que todos los navegadores deberían seguir para ser compatibles entre sí.

Tú podrás decidir cuando quieres que se registre el evento: en la fase de captura o en la fase de burbujeo. El tercer parámetro de `addEventListener` te permitirá indicar si lo haces en la **fase de captura (`true`)**, o en la **fase de burbujeo(`false`)**.

Por ejemplo:

```
elemento1.addEventListener('click',hacerAlgo1,true);
elemento2.addEventListener('click',hacerAlgo2,false);
```

Si el usuario hace click en el elemento2 ocurrirá lo siguiente:



1. El evento de click comenzará en la fase de captura. El evento comprueba si hay algún ancestro del elemento2 que tenga un evento de `onclick` para la fase de captura (`true`).
2. El evento encuentra un `elemento1.hacerAlgo1()` que ejecutará primero, pues está programado a `true`.
3. El evento viajará hacia el destino, pero no encontrará más eventos para la fase de captura. Entonces el evento pasa a la fase de burbujeo, y ejecuta `hacerAlgo2()`, el cual hemos registrado para la fase de burbujeo (`false`).
4. El evento viaja hacia arriba de nuevo y chequea si algún ancestro tiene programado un evento para la fase de burbujeo. Éste no será el caso, por lo que no hará nada más.

Para **detener la propagación del evento** en la fase de burbujeo, disponemos del método `stopPropagation()`. En la fase de captura es imposible detener la propagación.