

Tema 6:

Gestión de eventos y formularios en JavaScript.

Desarrollo Web en Entorno Cliente

Objetivos



- Capturar y gestionar los eventos producidos en una página web.
- Diferenciar los tipos de eventos que se pueden manejar.
- Crear código que capture y utilice eventos.
- Gestionar formularios web.
- Validar formularios web utilizando eventos y expresiones regulares.

Contenidos



1. Modelo de gestión de eventos.
2. Formularios.
3. Modificación de apariencia y comportamiento.
4. Validación y envío.
5. Expresiones regulares.



1.- Modelos de gestión de eventos

- Los eventos son mecanismos que se accionan cuando el usuario realiza un cambio sobre una página web.
- El encargado de crear la jerarquía de objetos que compone una página web es el DOM (*Document Object Model*).
- Por tanto es el DOM el encargado de gestionar los eventos.

1.- Modelos de gestión de eventos

- Para poder controlar un evento se necesita un manejador.
- Las funciones o código JavaScript que se definen para cada evento se denominan "*manejador de eventos*".
- En el caso del evento *click*, el manejador sería `onclick`.

— Ejemplo de manejador de eventos como atributo de un elemento html:

No es aconsejable(obsoleto)

```

```

1.- Modelos de gestión de eventos

- Ejemplo de manejador de eventos como función externa:

```
<html>
  <head>
    <title>Pagina de Evento</title>
    <script type="text/javascript">
      function func1() {
        alert("Click en imagen");
      }
    </script>
  </head>
  <body>
    
  </body>
</html>
```

1.- Modelos de gestión de eventos

- Ejemplo de manejador de eventos semánticos:

```
<html>
  <head>
    <title>Pagina de Evento</title>
    <script type="text/javascript">
      function func1() {
        alert("Click en imagen");
      }
      window.onload = function()
      {
        document.getElementById("imgworld").onclick
          =func1;
      }
    </script>
  </head>
  <body>
    
  </body>
</html>
```

1.- Modelos de gestión de eventos

- Modelo de registro de eventos tradicional:

```
function hacerAlgo() {
```

```
}
```

```
. . .
```

```
//para asignar un evento a un elemento  
elemento.onclick = hacerAlgo;
```

```
//para eliminar el gestor de evento  
del elemento  
elemento.onclick = null;
```


1.- Modelos de gestión de eventos

- Modelo de registro avanzado de eventos según W3C:
 - Para asignar un evento a un elemento:

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

- Este método tiene tres argumentos: el tipo de evento, la función a ejecutar y un valor booleano que se utiliza para indicar cuando se debe capturar el evento: en la fase de captura (true) o de burbujeo (false).

```
miDiv.addEventListener('click', muestraMensaje, false);
```

- Para desasociar la función de evento a un elemento:

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

```
miDiv.removeEventListener('click',muestraMensaje, false);
```

- Para cancelar un evento:

```
elemento.preventDefault;
```

Ejemplo anterior:

```
<A id="mienlace" href="pagina.html">Pulsa aquí </A>
<script type="text/javascript">
  document.getElementById("mienlace").onclick = alertar;
  function alertar()
  {
    alert("Te conectamos con la pagina: " + this.href);
  }
</script>
```

Modelo avanzado según W3C:

```
document.getElementById("mienlace").addEventListener(`cl
ic`,alertar, false);
```

La ventaja de este método es que podemos añadir tantos eventos como queramos:

```
document.getElementById("mienlace").addEventListener(`cl
ic`,alertar, false);
document.getElementById("mienlace").addEventListener(`cl
ic`,avisar, false);
document.getElementById("mienlace").addEventListener(`cl
ic`,chequear, false);
```

1.- Modelos de gestión de eventos

- Modelo de registro de eventos según Microsoft:
 - Para asignar un evento a un elemento:

```
elemento.attachEvent('evento', función)
```

- Este método tiene dos argumentos: el tipo de evento (el cual lleva en este caso el prefijo 'on', y la función a ejecutar.
- Los eventos siempre burbujan, no hay forma de captura.

```
miDiv.attachEvent('onclick', muestraMensaje);
```

- Para desasociar la función de evento a un elemento:

```
elemento.detachEvent('evento', función)
```

```
miDiv.detachEvent('onclick', muestraMensaje);
```



Comparando este modelo con el W3c encontramos dos diferencias importantes:

- Los eventos siempre burbujan, 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.)

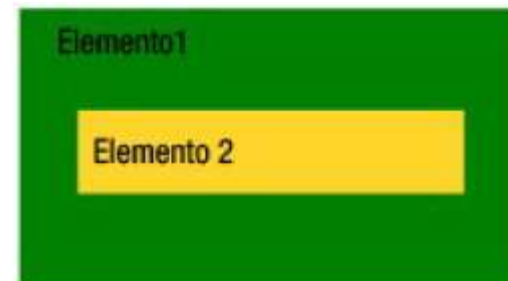
Listado de atributos de eventos (IE: Internet Explorer, F: Firefox, O: Opera, W3C: W3C Standard.)

Listado de atributos de eventos					
Atributo	El evento de produce cuando...	IE	F	O	W3C
onblur	Un elemento pierde el foco.	3	1	9	Si
onchange	El contenido de un campo cambia.	3	1	No	Si
onclick	Se hace clic con el ratón en un objeto.	3	1	9	Si
ondblclick	Se hace doble clic con el ratón sobre un objeto	4	1	9	Si
onerror	Ocurre algún error cargando un documento imagen.	4	1	9	Si
onfocus	Un elemento tiene el foco.	3	1	9	Si
onkeydown	Se precisa una tecla del teclado.	3	1	No	Si
onkeypress	Se presiona una tecla o se mantiene presionada.	3	1	9	Si
onkeyup	Cuando soltamos una tecla.	3	1	9	Si
onload	Una página o imagen terminaron de cargarse.	3	1	9	Si
onmousedown	Se presiona un botón del ratón.	4	1	9	Si
onmousemove	Se mueve el ratón.	3	1	9	Si
onmouseout	Movemos el ratón fuera de un elemento.	4	1	9	Si
onmouseover	El ratón se mueve sobre un elemento.	3	1	9	Si
onmouseup	Se libera un botón del ratón.	4	1	9	Si
onresize	Se redimensiona una ventana o frame.	4	1	9	Si
onselect	Se selecciona un texto.	3	1	9	Si
onunload	El usuario abandona una página.	3	1	9	Si

1.- Modelos de gestión de eventos

- **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? Dependerá del tipo de navegador que tengamos.
- Si el usuario hace click en el elemento2, provocará un click en ambos: elemento1 y elemento2, pero ¿cuál es el orden de los eventos?



1.- Modelos de gestión de eventos

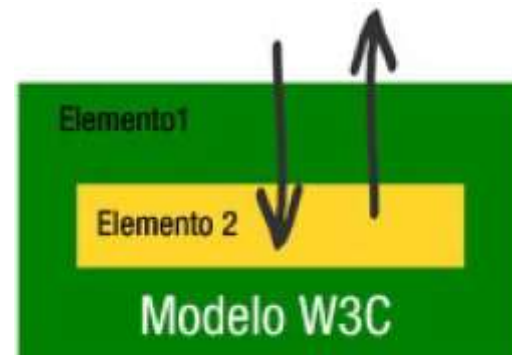
- **Orden de disparo de los eventos.**
 - Tenemos dos modelos propuestos por Nestcape y Microsoft en sus orígenes:
 - Nestcape: el evento en el elemento1 tendrá lugar primero. Es lo que se conoce como "**captura de eventos**".
 - Microsoft: el evento en el elemento2 tendrá precedencia. Es lo que se conoce como "**burbujeo de eventos**".



1.- Modelos de gestión de eventos

- **Orden de disparo de los eventos.**

- MODELO W3C: se decidió tomar una posición intermedia. Cuando se produce un evento, 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.
- El programador decidirá cuando quiere que se registre el evento: en la fase de captura o en la fase de burbujeo.



1.- Modelos de gestión de eventos

- Orden de disparo de los eventos.

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.

1.- Modelos de gestión de eventos

- **El objeto event**

- Cuando se produce un evento, no es suficiente con asignarle una función para procesar ese evento. Además se necesita información relativa al evento producido: la tecla que se ha pulsado, la posición del ratón, el elemento que ha producido el evento, etc.
- El objeto ***event*** es el mecanismo definido por los navegadores para proporcionar toda esa información. Se trata de un objeto que se crea automáticamente cuando se produce un evento y que se destruye de forma automática cuando se han ejecutado todas las funciones asignada al evento.
- Existen numerosas diferencias en cuanto a las propiedades y métodos del objeto ***event*** dependiendo del navegador utilizado. Consultar el anexo del objeto.



En los navegadores tipo Internet Explorer, el objeto event se obtiene directamente mediante:

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

En el resto de navegadores, se obtiene a partir del argumento que el navegador crea automáticamente:

```
function manejarEventos (evento)  
{  
    var evento = evento;  
}
```

Para programar una aplicacion que funcione en todos los navegadores:

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

Propiedades del objeto event

Propiedades del objeto evento	
Propiedad	Descripción
bubbles	Indica si el evento se propaga de manera ascendente por el DOM o no.
cancelable	Nos permite saber si el evento es cancelable. Un evento es cancelable si se puede detener, simulando que nunca ha ocurrido.
defaultPrevented	Toma valor true cuando el evento ha sido cancelado deliberadamente.
timeStamp	Nos dice el tiempo que ha transcurrido, en milisegundos, desde que se cargó la página hasta que se ha producido el evento.
type	Devuelve el tipo de evento que se ha producido como string ("click", "mouseup", "keydown", etc.)
target	Representa el elemento sobre el que se ha producido el evento.
currentTarget	Representa el elemento que ha lanzado el evento, es decir, el que tiene asociado el manejador.

Métodos del objeto event

Métodos del objeto evento	
Método	Descripción
preventDefault()	Cancela un evento, evitando así que se ejecute su acción por defecto. Esto no evita que el evento se siga propagando
stopPropagation()	Detiene la propagación de un evento hacia niveles superiores del DOM

Ejemplo:

```
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>
```

1.- Modelos de gestión de eventos

- **Incompatibilidades entre navegadores:**

- Crear páginas y aplicaciones web resulta más complejo de lo que debería debido a las incompatibilidades entre los navegadores.
- La incompatibilidad más importante se da precisamente en el modelo de eventos del navegador.
- Se ha de tener en cuenta las diferencias existentes entre los navegadores actuales para el desarrollo de aplicaciones web.



1.- Modelos de gestión de eventos

- La especificación DOM define cuatro grupos de eventos dividiéndolos según su origen:
 - Eventos del ratón.
 - Eventos del teclado.
 - Eventos HTML.
 - Eventos DOM.

1.- Modelos de gestión de eventos

- **Eventos del ratón (1):**

- click: este evento se produce cuando pulsamos sobre el botón principal (izquierdo) del ratón. El manejador de este evento es **onclick**.
- dblclick: este evento se acciona cuando hacemos un doble click sobre el botón principal del ratón. El manejador de este evento es **ondblclick**.
- mousedown: este evento se produce cuando pulsamos un botón del ratón. El manejador de este evento es **onmousedown**.
- mouseout: este evento se produce cuando el puntero del ratón esta dentro de un elemento y este puntero es desplazado fuera del elemento. El manejador de este evento es **onmouseout**.

1.- Modelos de gestión de eventos

- Eventos del ratón (2):
 - mouseover: este evento al revés que el anterior se produce cuando el puntero del ratón se encuentra fuera de un elemento, y este se desplaza hacia el interior. El manejador de este evento es **onmouseover**.
 - mouseup: este evento se produce cuando soltamos un botón del ratón que previamente teníamos pulsado. El manejador de este evento es **onmouseup**.
 - mousemove: se produce cuando el puntero del ratón se encuentra dentro de un elemento. Es importante señalar que este evento se producirá continuamente una vez tras otra mientras el puntero del ratón permanezca dentro del elemento. El manejador de este evento es **onmousemove**.

1.- Modelos de gestión de eventos

- Eventos del ratón:

- En los eventos del ratón hay propiedades comunes con la teclas. Por ejemplo, también tenemos propiedades **AltKey**, **CtrlKey**, **ShiftKey** y **MetaKey**, para saber si la pulsación de alguna de estas teclas acompaña al evento del ratón.
- Además, la propiedad **button** devuelve el botón del ratón pulsado en el momento del evento. Estos valores son:

Valor	Significado
0	Botón principal del ratón.
1	Botón central del ratón (en muchos casos la rueda)
2	Botón secundario del ratón.
3	Cuarto botón (retroceder página).
4	Quinto botón (avanzar página).

1.- Modelos de gestión de eventos

- Eventos del ratón:
 - Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente:
`mousedown, mouseup, click.`
 - Por tanto, la secuencia de eventos necesaria para llegar al `doubleclick` sería la siguiente:
`mousedown, mouseup, click,`
`mousedown, mouseup, click,`
`doubleclick.`

Actividad 1



Crea un elemento `<div>` y muestra, cuando el usuario pase el ratón por encima, el borde destacado. Cuando el ratón salga del div, se volverá a mostrar el borde original.

```
<div id="elemento" style="padding: .2em; width: 150px; height: 60px; border:
thin solid silver"
onmouseover = "document.getElementById('elemento').style.borderColor =
'blue'"
onmouseout = "document.getElementById('elemento').style.borderColor =
'silver'">
Sección de contenidos...
</div>
```

Actividad 1.1



Crea una página web que tenga un texto que indique que al pulsarla Alt+F12, podremos colocar una imagen de fondo. El texto tiene que salir centrado.

Inicialmente aparece una pantalla color agua marina con el texto y hasta que el usuario no pulse esa tecla, la imagen no se muestra.

Tras pulsar ALT+F12 una imagen ocupará el fondo completo.

Actividad 1.2



Crea una aplicación web que muestre una capa centrada que ocupe el 50% del ancho y el alto de la ventana.

Inicialmente la capa será blanca y solo se verá el borde.

Al arrimar el ratón se colorea en verde.

Al hacer clic encima con el botón principal se colorea de rojo, pero solo mientras el botón principal esta abajo, sino se quite en color rojo.

Al hacer clic encima con el botón secundario ocurre lo mismo que en el caso anterior, pero se muestra el color azul. No se mostrara en ningún caso el menú de contexto.

1.- Modelos de gestión de eventos

- Eventos del teclado:
 - keydown: este evento se produce cuando pulsamos cualquier tecla del teclado. Si mantenemos pulsada una tecla de forma continua, el evento se produce una y otra vez hasta que soltemos la misma. El manejador de este evento es **onkeydown**.
 - keypress: este evento se produce si pulsamos una tecla de un carácter alfanumérico. (El evento no se produce si pulsamos enter, la barra espaciadora, etc...). En el caso de mantener una tecla pulsada, el evento se produce de forma continuada. El manejador de este evento es **onkeypress**.
 - keyup: este evento se produce cuando soltamos una tecla. El manejador de este evento es **onkeyup**.

1.- Modelos de gestión de eventos

- Eventos del teclado:
 - Cuando se pulsa una tecla correspondiente a un *carácter alfanumérico*, se produce la siguiente secuencia de eventos:
`keydown`, `keypress`, `keyup`.
 - Cuando se pulsa otro tipo de tecla, la secuencia es:
`keydown`, `keyup`.
 - Si se mantiene pulsada la tecla, para el primer caso (caracteres alfanuméricos) se repiten de forma continua los eventos: `keydown` y `keypress` y para el segundo caso, se repite el evento `keydown` de forma continua.

Actividad 2



Utiliza el evento adecuado para mostrar un aviso de que «*se ha soltado la tecla que has pulsado*» para cualquier tecla del teclado.

1.- Modelos de gestión de eventos

- Eventos HTML (1):

- load: el evento *load* hace referencia a la carga de distintas partes de la página. Este se produce en el objeto `Window` cuando la página se ha cargado por completo. En el elemento `` actúa cuando la imagen se ha cargado. En el elemento `<object>` se acciona al cargar el objeto completo. El manejador es **onload**.
- unload: el evento *unload* actúa sobre el objeto `Window` cuando la pagina ha desaparecido por completo (por ejemplo, si pulsamos el aspa cerrando la ventana del navegador). También se acciona en el elemento `<object>` cuando desaparece el objeto. El manejador es **onunload**.
- abort: este evento se produce cuando el usuario detiene la descarga de un elemento antes de que haya terminado, actúa sobre un elemento `<object>`. El manejador es **onabort**.

1.- Modelos de gestión de eventos

- Eventos HTML (2):

- error: el evento *error* se produce en el objeto `Window` cuando se ha producido un error en JavaScript. En el elemento `` cuando la imagen no se ha podido cargar por completo y en el elemento `<object>` en el caso de que un elemento no se haya cargado correctamente. El manejador es **onerror**.
- select: se acciona cuando seleccionamos texto de los cuadros de textos `<input>` y `<textarea>`. El manejador es **onselect**.
- change: este evento se produce cuando los cuadros de texto `<input>` y `<textarea>` pierden el foco y el contenido que tenían ha variado. También se producen cuando un elemento `<select>` cambia de valor. El manejador es **onchange**.
- submit: este evento se produce cuando pulsamos sobre un botón de tipo submit. El manejador es **onsubmit**.

1.- Modelos de gestión de eventos

- Eventos HTML (3):
 - reset: este evento se produce cuando pulsamos sobre un botón de tipo reset. El manejador es **onreset**.
 - resize: este evento se produce cuando redimensionamos el navegador, actúa sobre el objeto `window`. El manejador es **onresize**.
 - scroll: se produce cuando varía la posición de la barra de scroll en cualquier elemento que la tenga. El manejador es **onscroll**.
 - focus: este evento se produce cuando un elemento obtiene el foco. El manejador es **onfocus**.
 - blur: este evento se produce cuando un elemento pierde el foco. El manejador es **onblur**.

1.- Modelos de gestión de eventos

- Eventos DOM:
 - **DOMSubtreeModified.** Este evento se produce cuando añadimos o eliminamos nodos en el subárbol de un elemento o documento.
 - **DOMNodeInserted.** Este evento se produce cuando añadimos un nodo hijo a un nodo padre.
 - **DOMNodeRemoved.** Este evento se produce cuando eliminamos un nodo que tiene nodo padre.
 - **DOMNodeRemovedFromDocument.** Este evento se produce cuando eliminamos un nodo del documento.
 - **DOMNodeInsertedIntoDocument.** Este evento se produce cuando añadimos un nodo al documento.

1.- Modelos de gestión de eventos

- Eventos de movimientos en la ventana:

Eventos de movimiento en la ventana	
evento	Descripción
scroll	Ocurre cuando se ha desplazado la ventana a través de las barras de desplazamiento o usando el dispositivo táctil.
resize	Evento de <code>window</code> que se produce cuando se cambia el tamaño de la ventana.

- Eventos sobre carga y descarga de elementos:

Eventos sobre carga y descarga de elementos	
evento	Descripción
load	Se concluyó la carga del elemento. Es uno de los elementos más importantes a capturar para asegurar que el código siguiente funciona con la seguridad de que está cargado cuando se necesita. Cuando se aplica al elemento <code>window</code> , se produce cuando todos los elementos del documento se han cargado.
DOMContentLoaded	Similar al evento <code>load</code> . Se produce cuando el documento HTML ha sido cargado. A diferencia de <code>load</code> , se dispara sin esperar a que se terminen de cargar las hojas de estilos, imágenes y elementos en segundo plano. En general, para JavaScript, es más conveniente este evento.
abort	Se produce cuando se anula la carga de un elemento.
error	Sucede si hubo un error en la carga.
progress	Se produce si la carga está en proceso.
readystatechange	Ocurre cuando se ha modificado el estado del atributo <code>readystatechange</code> , lo cual ocurre cuando se ha modificado el estado de carga y descarga.

1.- Modelos de gestión de eventos

- Eventos sobre el historial:

Eventos sobre el historial	
evento	Descripción
popstate	Se produce si se cambia el historial.

- Eventos relacionados con la reproducción de medios:

Eventos relacionados con la reproducción de medios	
evento	Descripción
waiting	Sucede cuando el video se ha detenido por falta de datos.
playing	El medio está listo para su reproducción después de que se detuviera por falta de datos u otras causas.
canplay	Ocurre cuando se detecta que un video (u otro elemento multimedia) ya se puede reproducir (aunque no se haya cargado del todo)
canplaythrough	Se produce si se estima que el video ha cargado suficientes datos para poderse reproducir sin tener que esperar la llegada de más datos.
pause	El medio de reproducción se ha pausado.
play	Ocurre cuando el medio de reproducción se ha empezado a reproducir tras una pausa.
ended	Se produce si la reproducción del video o audio se ha detenido, sea por haber llegado al final o porque no hay más datos disponibles.
loadeddata	Se produce si se ha cargado el frame actual.
suspend	Se lanza si se ha suspendido la carga del medio.
emptied	Sucede si se ha vaciado el medio por un nuevo intento de carga por parte del usuario o por otras razones.
stalled	Sucede ante un fallo en la carga, pero sin que se detenga la misma.
seeking	Ocurre cuando se ha iniciado una labor de búsqueda en el medio.
seeked	Ocurre cuando se ha finalizado la labor de búsqueda.
loadedmetadata	Se han cargado los metadatos del medio.
durationchange	Sucede si se modifica el atributo duration del medio.
timeupdate	Ocurre si se ha modificado el atributo currentTime del medio.
ratechange	Se produce cuando el ratio del video se ha modificado.
volumechange	Se lanza si el volumen se ha modificado.

1.- Modelos de gestión de eventos

- Eventos de arrastre:

Están relacionados con la interface que es parte de HTML5. Para que un elemento pueda ser arrastrable debe tener el atributo **draggable** colocado a valor **true**:

```
<div id="capaArrastrable" draggable="true"> ¡Soy arrastrable! </div>
```

En este tipo de operaciones hay dos protagonistas: una capa arrastrable (la que realmente se arrastra) y un posible destino del arrastre.

1.- Modelos de gestión de eventos

- Eventos de arrastre:

Eventos de arrastre	
evento	Descripción
dragstart	Se produce cuando el usuario empieza a arrastrar el elemento.
drag	Ocorre una vez iniciado el arrastre, cada vez que se sigue arrastrando el elemento.
dragstop	Se produce cuando el arrastre finaliza.

Eventos que se producen en el elemento destino del arrastre:

Eventos de arrastre	
evento	Descripción
dragenter	Se produce cuando el elemento que se está arrastrando, entra en el elemento destino.
dragover	Ocorre cada vez que se continua, tras haber entrado, arrastrando el elemento origen sobre el destino.
dragleave	Ocorre cuando el elemento que se arrastra, sale del destino.
drop	Ocorre cuando el elemento origen se suelta dentro del destino. Para que ese evento se pueda capturar hay que eliminar el comportamiento por defecto del evento dragover .

1.- Modelos de gestión de eventos

- Eventos sobre animación y transiciones:

Eventos sobre animaciones y transiciones	
evento	Descripción
animationstart	Se produce cuando se inicia una animación sobre el elemento.
animationinteraction	Se produce justo cuando se repite la animación.
animationend	Se produce cuando finaliza a animación.
transitionrun	Se lanza cuando ya se ha preparado para empezar la transición.
transitionstart	Ocurre cuando se inicia una transición.
transitionend	Ocurre al finalizar la transición.

- Eventos del portapapeles:

Eventos del portapapeles	
evento	Descripción
cut	Se produce cuando el usuario intenta cortar contenido del elemento.
copy	Se produce cuando el usuario intenta copiar contenido del elemento.
paste	Se produce cuando el usuario intenta pegar contenido.

1.- Modelos de gestión de eventos

- Eventos especiales:

Eventos especiales	
evento	Descripción
offline	Solo funciona para el objeto <code>window</code> y se produce si el navegador se desconecta de la red.
online	Solo funciona para <code>window</code> y se produce si el navegador vuelve a conectarse a la red después de haber estado desconectado.
fullscreenchange	Ocurre cuando un elemento pasa a modo de pantalla completa.
fullscreenerror	Sucede si hay error al pasar un elemento a modo pantalla completa.
message	Evento que se asocia a numerosos elementos de envío de mensajes como los que se producen a través de las APIs <code>WebSockets</code> , <code>webWorkers</code> y otras.

Actividad 3



Resalta el campo que está activo en cada momento. Es decir, cuando el usuario se encuentre el campo nombre, el borde debe ser mas grueso y de color azul. Cuando el usuario pase a otro campo (pierda el foco), vuelva a su borde original.



Escribe tus datos personales

Nombre:

E-mail:

2.- Formularios

- Un formulario web sirve para enviar, tratar y recuperar datos que son enviados y recibidos entre un cliente y un servidor web.
- En JavaScript el objeto form depende en la jerarquía del objeto document.
- JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios.