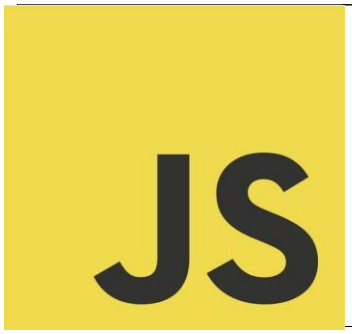

Desarrollo web en entorno cliente

TEMA – 5

Eventos

ÍNDICE

1. INTRODUCCIÓN

Los eventos son manejadores que nos proporciona el navegador para que cuando detecte que se produzca una acción (evento), se ejecute un código asociado a esa acción. En esta unidad trataremos los eventos existentes en Javascript y las distintas formas de manejarlos.

2. EVENTOS

Para poder controlar un evento se necesita un manejador. Básicamente, el manejador es la palabra reservada que indica la acción que va a manejar.

En el caso del evento click , el manejador sería onClick.

```
<IMG SRC="mundo.jpg" onclick="alert('Click en imagen');">
```

2.1 Principales eventos

A continuación mostramos un listado de los principales eventos existentes en Javascript:

- **onfocus**
  Al obtener un foco.
- **onblur**
  Al salir del foco de un elemento.
- **onchange**
  Al hacer un cambio en un elemento.
- **onclick**
  Al hacer un click en el elemento.
- **onkeydown**
  Al pulsar una tecla (sin soltarla).
- **onkeyup**
  Al soltar una tecla pulsada.
- **onkeypress**
  Al pulsar una tecla.
- **onload**
  Al cargarse una página.
- **onunload**
  Al descargarse una página (salir de ella).
- **onmousedown**
  Al hacer clic de ratón (sin soltarlo).
- **onmouseup**
  Al soltar el botón del ratón previamente pulsado.
- **onmouseover**
  Al entrar encima de un elemento con el ratón.
- **onmouseout**
  Al salir de encima de un elemento con el ratón.
- **onsubmit**
  Al enviar los datos de un formulario.
- **onreset**
  Al resetear los datos de un formulario.
- **onselect**
  Al seleccionar un texto.
- **onresize**
  Al modificar el tamaño de la página del navegador.

El total de eventos disponibles está descrito en
http://www.w3schools.com/jsref/dom_obj_event.asp

3. MODELOS DE GESTIÓN DE EVENTOS

3.1 Manejo de eventos desde elementos XHTML

La forma más sencilla (aunque menos práctica para tener un código limpio y ordenado) de indicar que hay un evento asociado a un elemento XHTML es indicándolo en el propio código.

```
<input type="button" value="Boton Hola mundo"
     onclick="alert('Hola mundo');alert('Adios');"
/>
```

También en lugar de ejecutar una serie de instrucciones, es posible llamar a una función predefinida.

```
<input type="button" value="Boton miFuncion"
     onclick="miFuncion('cadenaParam1');"
/>
```

3.2 Asignación de eventos desde código a objetos XHTML

Podemos asignar/modificar mediante código el manejador de un evento predefinido en un objeto XHTML de una forma similar a esta. Supongamos que tenemos un objeto con id=”miObjeto” que posee el evento “onclick” sin asignar y la función “mostrarMensaje”.

Podemos referenciar al elemento XHTML y asignar la función como manejador del evento como vemos en este código

```
function mostrarMensaje(){
     alert(“Hola”);
}
document.getElementById(“miObjeto”).onclick=mostrarMensaje;
```

Atención: fijaros que pone “mostrarMensaje”. Así asigna la función como manejadora del evento. Si ponemos “mostrarMensaje()” no funcionará, ya que ejecutará la función y asignará su resultado al manejador.

3.3 Declarando eventos desde código

Podemos declarar eventos mediante código usando **addEventListener**(evento,manejador).

```
function mostrarMensaje(evento){ if(evento.type==”keyup”){
              alert(evento.keyCode);
     }
     else if(evento.type==”click”){ alert(evento.clientX+” “+evento.clientY);
     }
}
document.getElementById(“miObjeto”).addEventListener(“click”,mostrarMensaje);
document.addEventListener(“keyup”,mostrarMensaje);
document.getElementById(“miObjeto”).addEventListener(“dblclick”,function (){
 alert(“Codigo metido directamente”);
});
```

4. TIPOS DE EVENTOS

La especificación DOM define cuatro grupos de eventos dividiéndolos según su origen:

- Eventos del ratón.
- Eventos del teclado.
- Eventos HTML.
- Eventos DOM.
- Eventos FORM

4.1 Eventos del ratón

Los eventos que se incluyen en esta clasificación son los siguientes:

- **Click** → Este evento se produce cuando pulsamos sobre el botón izquierdo del ratón. El **manejador** de este evento es **onclick**.
- **Dblclick** → Este evento se acciona cuando hacemos un doble click sobre el botón izquierdo 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**.
- **Mouseup** → Este evento se produce cuando soltamos un botón del ratón que previamente teníamos pulsado. El **manejador** de este evento es **onmouseup**.
- **Mouseover** → Este evento se produce cuando el puntero del ratón se encuentra fuera de un elemento, y este se desplaza hacia el interior del elemento o alguno de sus hijos. El **manejador** de este evento es **onmouseover**.
- **Mouseout** → Este evento se produce cuando el puntero del ratón está dentro de un elemento y este puntero es desplazado fuera del elemento o alguno de sus hijos. El **manejador** de este evento es **onmouseout**.
- **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**.
- **Mouseenter** → Este evento se produce cuando el puntero del ratón entra dentro de un elemento. El **manejador** de este evento es **onmouseenter**.
- **Mouseleave** → Este evento al revés que el anterior se produce cuando el puntero del ratón sale fuera de un elemento. El **manejador** de este evento es **onmouseleave**.
- **Contextmenu** → Ocurre al pulsar con el botón derecho del ratón. El manejador de este evento es **oncontextmenu**
- **wheel** → Ocurre al mover la rueda del ratón. El **manejador** de este evento es **onwheel**.

Orden de ejecución de los eventos del ratón:

- Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente: mousedown, mouseup, click
- La secuencia de eventos necesaria para llegar al doble click llega a ser tan compleja como la siguiente: mousedown, mouseup, click, mousedown, mouseup, click, dblclick.

```
<body>
    <img
      onmouseover="zoom(this)"
      nmouseout="normal(this)"
      border="0" src="imgs/js.png" alt="JSLogo" width="32" height="32">
</body>
```

4.2 Eventos del teclado

Los eventos que se incluyen en esta clasificación son los siguientes:

- **Keydown** → Este evento se produce cuando pulsamos una 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**.
- **Keyup** → Este evento se produce cuando soltamos una tecla. El **manejador** de este evento es **onkeyup**.
- **Keypress** → Este evento se produce si pulsamos una tecla de un carácter alfanumérico (El evento no se produce si pulsamos shift, shift , control, alt alt, , alt gr). En el caso de mantener una tecla pulsada, el evento se produce de forma continuada. El **manejador** de este evento es **onkeypress**.

El objeto event contiene las siguientes **propiedades** para los eventos de teclado:

- **altKey** → Devuelve true si la tecla alt se ha pulsado
- **code** → Devuelve el código asociado a la tecla pulsada
- **ctrlKey** → Devuelve true si la tecla control se ha pulsado
- **key** → Devuelve la tecla pulsada
- **shiftKey** → Devuelve true si la tecla shift se ha pulsado

4.3 Eventos HTML

Los eventos que se incluyen en esta clasificación son los siguientes:

- **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 <img> 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**.
- **Error**. El evento error se produce en el objeto Window cuando se ha producido un error en JavaScript. En el elemento <img> 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**.
- **Beforeunload** → Este evento se produce cuando se pulsa un enlace, refrescamos, cerramos,...El **manejador** es **onbeforeunload**.
- **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**.

```
window.addEventListener("load", function (event) {
    console.log(se ha cargado la página al completo);
},false);
```

4.4 Eventos DOM

Los eventos que se incluyen en esta clasificación son los siguientes:

- **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.

```
var textNode = document . createTextNode ( "Mi texto" );
textNode.addEventListener('DOMNodeInserted',OnNodeInserted, false );
función OnNodeInserted ( evento ) {
    var textNode = evento . objetivo ;
    alert ( "El nodo de texto "' + textNode. data + "' ha sido agregado a un elemento." );
}
```

5. GESTIÓN DE EVENTOS

5.1 Uso del objeto this

Cuando ejecutas código dentro de un evento, existe un objeto llamado **“this”.**
Este objeto es una referencia al elemento que se ha producido el evento. Ejemplo, si el evento se ha producido un evento al hacer clic a una imagen con id=”miImagen”, el objeto “this” será lo mismo que poner “document.getElementById(“miImagen”);

```
<div id="cont" style="width:150px; height:60px; border:thin solid
silver"
onmouseover=this.style.borderColor='black';"
onmouseout="this.style.borderColor='red';">
  Contenidos
</div>
```

5.2 Obteniendo información del objeto event

Cuando se crea una función como manejador, al producirse el evento el navegador automáticamente manda como parámetro un objeto de tipo event.

```
function mostrarMensaje(evento){
     alert(evento.type);
}
document.getElementById(“miObjeto”).onclick=mostrarMensaje;
```

Este objeto posee cierta información útil del evento que se ha producido.

- **type**: dice el tipo de evento que es (“click”, “mouseover”, etc...). Devuelve el nombre del evento tal cual, sin el “on”. Es útil para hacer una función que maneje varios eventos.
- **keyCode**: en eventos de teclado, almacena el código de tecla de la tecla afectada por el evento.
- **clientX** / clientY: en eventos del ratón, devuelve las coordenadas X e Y donde se encontraba el ratón, tomando como referencia al navegador.
- **screenX** / screenY: en eventos del ratón, devuelve las coordenadas X e Y donde se encontraba el ratón, tomando como referencia la pantalla del ordenador.

```
function mostrarMensaje(evento){
     if(evento.type==”keyup”){
          alert(evento.keyCode);
     }
     else if(evento.type==”click”){
          alert(evento.clientX+” “+evento.clientY);
     }
}
document.getElementById(“miObjeto”).onclick=mostrarMensaje;
document.onkeyup=mostrarMensaje;
```

5.3 Ejemplo Ampliado

```
<input id="idBoton" type="button"  onclick="decirHola(this)" />                    // uso THIS
function decirHola(elemento) {
        alert("Has pulsado : " + elemento.id );
}

<input id="idBoton" type="button"  onclick="decirHola(event)" />                   // uso EVENT
function decirHola(e) {
        alert("elemento pulsado : " + e.target.textContent);
}

document.getElementById("idhref").addEventListener("click", decirHola);           // uso EVENT y THIS
function decirHola(e){
        alert("elemento pulsado : " + e.target.textContent);
        alert("elemento pulsado : " + this.textContent);
        e.preventDefault();
}
```

De los usos principales de JavaScript está la implementación de algoritmos que reaccionan a eventos que se producen en una página web. Cuando se termina de cargar completamente una página web se dispara el evento onload.

A lo largo de la historia de JavaScript, se han implemantado formas distintas de capturar los eventos que emite el navegador. Vamos a ver algunas mas frecuentes:

Eventos definidos accediendo a propiedades del objeto.

Esta metodología es todavía ampliamente utilizada ya que la mayoría de los navegadores lo implementan en forma similar.

Problema

Implementar un formulario que solicite la carga del nombre de usuario y su clave. Mostrar un mensaje si no se ingresan datos en alguno de los dos controles.

```
<body>

    <form method="post" action="procesar.php" id="formulario1">
        Ingrese nombre:
        <input type="text" id="usuario" name="usuario" size="20">
        <br> Ingrese clave:
        <input type="password" id="clave" name="clave" size="20">

        <br>
        <input type="submit" id="confirmar" name="confirmar" value="Confirmar">
    </form>

    <script>
        window.onload = inicio;
        function inicio() {
            document.getElementById("formulario1").onsubmit = validar;
        }
        function validar() {
            let usu = document.getElementById("usuario").value;
            let cla = document.getElementById("clave").value;
            if (usu.length == 0 || cla.length == 0) {
                alert('El nombre de usuario o clave está vacío');
                return false;
            } else
               return true;
        }
   </script>
</body>
```

Con esta metodología vemos que el código HTML queda limpio de llamadas a funciones JavaScript y todo el código queda dentro del bloque del script (pudiendo luego llevar todo este bloque a un archivo externo *.js)

Lo primero que vemos es inicializar la propiedad onload del objeto window con el nombre de la función que se ejecutará cuando finalice la carga completa de la página, es importante notar que a la propiedad onload le asignamos el nombre de la función y NO debemos disponer los paréntesis abiertos y cerrados (ya que no se está llamando a la función sino le estamos pasando la dirección o referencia de la misma)

```
        window.onload = inicio;
```

La función inicio es llamada por el objeto window cuando se termina de cargar la página. En esta función obtenemos la referencia del objeto formulario1 mediante el método getElementById e inicializamos la propiedad onsubmit con el nombre de la función que será llamada cuando se presione el botón submit del formulario:

```
        function inicio() {
            document.getElementById("formulario1").onsubmit = validar;        }
```

Por último tenemos la función validar que verifica si los dos controles del formulario están cargados:

```
function validar() {
    let usu = document.getElementById("usuario").value;
    let cla = document.getElementById("clave").value;
    if (usu.length == 0 || cla.length == 0) {
        alert('El nombre de usuario o clave está vacío');
        return false;
    } else
        return true;
}
```

La misma metodología pero utilizando funciones anónimas para cada evento el código ahora queda mas conciso:

```
<body>

    <form method="post" action="procesar.php" id="formulario1">
       Ingrese nombre:

         <input type="text" id="usuario" name="usuario" size="20">
        <br> Ingrese clave:
        <input type="password" id="clave" name="clave" size="20">
        <br>
        <input type="submit" id="confirmar" name="confirmar" value="Confirmar">
    </form>

    <script>
        window.onload = function() {
            document.getElementById("formulario1").onsubmit = function() {
               let usu = document.getElementById("usuario").value;
                let cla = document.getElementById("clave").value;
               if (usu.length == 0 || cla.length == 0) {
                    alert('El nombre de usuario o clave está vacío');
                   return false;
                } else
                    return true;
            }
       }
   </script>
</body>
```

a la propiedad onload del objeto window le asignamos una función anónima:

```
window.onload = function() {
...
}
```

En la implementación de la función anónima inicializamos la propiedad onsubmit del objeto formulario1 con otra función anónima:

```
document.getElementById("formulario1").onsubmit = function() {
...
}
```

Esta sintaxis de funciones anónimas es ampliamente utilizado.

Modelo de eventos definidos por W3C (World Wide Web Consortium)

Este modelo de eventos se basa en la implementación de un método para todos los objetos. La sintaxis es:

```
addEventListener(evento, método a ejecutar);
```

Veamos como implementamos el problema anterior utilizando este nuevo modelo de eventos:

```
<body>
    <form method="post" action="procesar.php" id="formulario1">
        Ingrese nombre:
        <input type="text" id="usuario" name="usuario" size="20">
        <br> Ingrese clave:
       <input type="password" id="clave" name="clave" size="20">
        <br>
        <input type="submit" id="confirmar" name="confirmar" value="Confirmar">
    </form>
    <script>
        window.addEventListener('load', inicio);
        function inicio() {
            document.getElementById("formulario1").addEventListener('submit',
validar);
       }
        function validar(evt) {
            let usu = document.getElementById("usuario").value;
           let cla = document.getElementById("clave").value;
            if (usu.length == 0 || cla.length == 0) {
               alert('El nombre de usuario o clave está vacío');
                evt.preventDefault();
            }
        }
   </script>
</body>
```

Lo primero que vemos es que en vez de inicializar la propiedad onload procedemos a llamar al método addEventListener:

```
window.addEventListener('load', inicio);
```

El primer parámetro es un string con el nombre del evento a inicializar, el segundo parámetro es el nombre de la función a ejecutar .

Cuando se carga completamente la página el objeto window tiene la referencia al método que se debe llamar, en nuestro caso se llama inicio. La función inicio obtiene la referencia del objeto formulario1 y procede a registrar el evento submit indicando en el segundo parámetro el nombre de la función que debe ejecutarse:

```
function inicio() {
    document.getElementById("formulario1").addEventListener('submit', validar);
}
```

El código de la función validar se modifica, llega como parámetro una referencia al evento y mediante este llamamos al método preventDefault si queremos que no se envíen los datos al servidor:

```
function validar(evt) {
    let usu = document.getElementById("usuario").value;
    let cla = document.getElementById("clave").value;
    if (usu.length == 0 || cla.length == 0) {
        alert('El nombre de usuario o clave está vacío');
        evt.preventDefault();
    }
}
```

Este modelo de eventos está siendo ampliamente implementado por los navegadores modernos. El problema es el IE8 o inferiores que no lo implementa de esta forma.

Evento DOMContentLoaded

- Evento 'load': El evento load se dispara cuando el contenido del archivo HTML y las referencias a todos los recursos asociados (imágenes, hojas de estilo etc.) se han cargado en memoria del navegador.
- Evento 'DOMContentLoaded': El evento DOMContentLoaded se dispara cuando el contenido del archivo HTML se ha cargado en el navegador, sin necesitar esperar imágenes, hojas de estilo etc.
  Es muy conveniente en la mayoría de las veces inicializar script de JavaScript utilizando éste evento en lugar del evento 'load'. Todos los navegadores modernos disponen del evento 'DOMContentLoaded'.

En el problema anterior solo cambiamos la referencia del evento load por DOMContentLoaded:

```
<body>

    <form method="post" action="procesar.php" id="formulario1">
       Ingrese nombre:
        <input type="text" id="usuario" name="usuario" size="20">
       <br> Ingrese clave:
        <input type="password" id="clave" name="clave" size="20">
        <br>
        <input type="submit" id="confirmar" name="confirmar" value="Confirmar">
   </form>
    <script>
        window.addEventListener('DOMContentLoaded', inicio);
        function inicio() {
            document.getElementById("formulario1").addEventListener('submit',
validar);
        }

        function validar(evt) {
            let usu = document.getElementById("usuario").value;
            let cla = document.getElementById("clave").value;
            if (usu.length == 0 || cla.length == 0) {
                alert('El nombre de usuario o clave está vacío');
                evt.preventDefault();
            }
        }
   </script>
</body>
```