JAVASCRIPT

CLASE 9

Material complementario

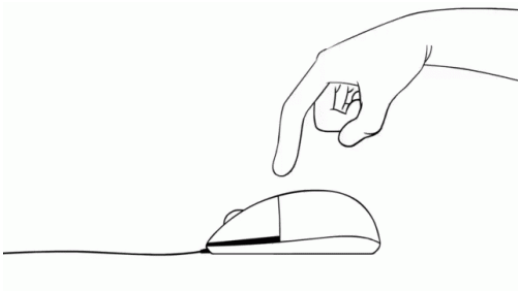
EVENTOS

CODER HOUSE

JavaScript: eventos

Evento: definición

Los eventos son la manera que tenemos en Javascript de controlar las acciones del usuario, y definir el comportamiento de la aplicación en respuesta a las mismas. Supongamos que el usuario hace clic sobre un botón de la interfaz: a esta situación se la identifica como ***evento***.



Elementos del DOM y el navegador cuentan con ***eventos asociados***, los cuales podemos utilizar para establecer la interpretación de un bloque de instrucciones cuando ocurren. El tiempo en el que un evento pasa (o "se dispara", como se suele decir) puede variar, ya que en la mayoría de los casos depende de un comportamiento del usuario sobre la interfaz. Por dicha razón, la respuesta a un evento, la cual se define en forma de función y se la suele llamar manejador de eventos o "event handlers", se interpreta automáticamente cuando el evento ocurre.

Si bien un elemento del DOM cuenta con un conjunto de eventos posibles, es responsabilidad del programador/ra definir qué eventos se va a controlar en la página con JavaScript. Esto se realiza asociando un manejador de eventos al evento escogido, procedimiento que cuenta con tres alternativas de declaración que desarrollaremos en los siguientes párrafos.

Definiendo eventos con JavaScript

Para determinar en el script qué eventos vamos a controlar en la página web (también se puede decir “eventos a escuchar en la página web”), es necesario emplear una de las tres notaciones presentadas a continuación.

Cabe aclarar que, por fines explicativos, el código JavaScript se encuentra en el documento HTML, entre etiquetas <script></script>, pero a esta altura del curso se recomienda codificar la lógica de la aplicación siempre en un archivo .js aparte. Habiendo dicho esto, analicemos las tres notaciones disponibles para determinar el control de un evento. Con el objetivo de percibir diferencias de declaración, en todos los ejemplos *se escucha* el evento clic sobre un botón.

Opción 1: método addEventListener

El método addEventListener permite definir qué evento escuchar sobre cualquier elemento del DOM. El primer parámetro corresponde al nombre del evento, y el segundo a la función de respuesta. El manejador de eventos puede ser una función con nombre, anónima o función flecha. Veamos ahora un ejemplo de codificación de esta opción:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primer App</title>
  </head>
  <body>
    <h2>Coder House</h2>
    <button id="btnPrincipal">CLICK</button>
    <script>
      let boton = document.getElementById("btnPrincipal")
      boton.addEventListener("click", respuestaClick)
      function respuestaClick(){
        console.log("Respuesta evento");
```

```
      }
    </script>
  </body>
</html>
```

Como podemos observar, primero obtenemos el elemento sobre el cual queremos definir el evento; en este adquirimos la referencia al botón empleando el método ***getElementById***. Realizada dicha instrucción, podemos emplear la variables botón para llamar al método ***addEventListener***, enviando los parámetros necesarios.

Gracias a estas instrucciones, cada vez que el usuario haga un clic en el botón cuyo id es ***btnPrincipal,*** se efectuará una salida por consola con el mensaje "Respuesta evento".

Opción 2: propiedades on-event

Podemos emplear una propiedad on-event para asignar el manejador de evento, definiendo la respuesta al evento escogido. Para reconocer el identificador de la propiedad, utilizamos el nombre del evento y el prefijo ***on***, siendo en el caso del evento clic la propiedad ***onclick***. Veamos el ejemplo correspondiente:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primer App</title>
  </head>
  <body>
    <h2>Coder House</h2>
    <button id="btnPrincipal">CLICK</button>
    <script>
      let boton = document.getElementById("btnPrincipal")
      boton.onclick = () =>{console.log("Respuesta 2")}
    </script>
  </body>
</html>
```

En la opción 2, también obtenemos el elemento sobre el cual queremos definir el evento, pero en este caso utilizamos el acceso por propiedad ***boton.onclick,*** y asignamos una función flecha como manejador. Gracias a estas instrucciones, cada vez que el usuario realice un clic en el botón cuyo id es ***btnPrincipal*** se efectuará una salida por consola con el mensaje "Respuesta 2".

Opción 3: atributo HTML

La última opción es determinar el evento a escuchar especificando el manejador del evento en el atributo de una etiqueta HTML. La denominación del atributo es idéntica al de la propiedad ***on-event:***

```
<input type="button" value="CLICK2" onclick="alert('Respuesta 3');" />
```

¿Qué opción elegir?

Se recomienda emplear el método addEventListener o propiedades on-event para determinar los eventos a escuchar. Las podemos identificar como formas de declaración equivalentes, aunque pueden existir casos de aplicación específica: por ejemplo, al usar la opción 1 el nombre del evento puede venir de una variable, cosa que no puede hacerse en la segunda al usar la propiedad.

La opción 3, aunque es de fácil implementación, no es recomendada para proyectos en producción, ya que no se considera una buena práctica declarar funciones y código JavaScript dentro del HTML.

Tipos de eventos

Si bien ya sabemos como definir los eventos a escuchar en la apliacaciòn, determinar cuál utilizar comprende reconocer bajo qué condiciones se dispara cada evento. Esto

puede ser una tarea inicial un tanto extenuante, ya que son bastantes los eventos estándar (que funcionan en todos los navegadores) definidos en las especificaciones oficiales, Por ende, se recomienda empezar con los eventos que son más empleados en el desarrollo web, entre los cuales distinguimos distintos tipos:

Eventos del mouse

Los eventos del mouse, llamados MouseEvent, son aquellos que se producen por las acciones del usuario con el mouse. De los eventos en esta categoría, los más utilizados son:

- Mousedown: se dispara cuando se oprime un botón del ratón sobre un elemento.
- Mouseup: se dispara cuando se suelta un botón del ratón sobre un elemento.
- Mouseover: se dispara cuando el puntero del mouse se posiciona sobre un elemento.
- Mouseout: el puntero del mouse se sale del elemento.
- Mousemove: el movimiento del mouse sobre el elemento activa el evento.
- Clic: se activa al disparar el evento mousedown o mouseup sobre un elemento.

Veamos ahora un ejemplo de definición de evento de tipo mouse:

```
//CODIGO HTML DE REFERENCIA
<button id="btnMain">CLICK</button>
//CODIGO JS
let boton          = document.getElementById("btnMain");
boton.onclick      = () => {console.log("Click")};
boton.onmousemove  = () => {console.log("Move")}
```

Eventos del teclado

Los eventos de teclado o KeyboardEvent, ocurren ante una interacción del usuario con el teclado. Este tipo de eventos permite determinar cuándo el usuario presiona una tecla, siendo muy útil para controlar entradas que el mismo haga sobre etiquetas input. De

este tipo de eventos destacamos:

- Keydown: se dispara cuando se presiona una tecla sobre el elemento.
- Keyup: se dispara cuando se suelta una tecla sobre el elemento.

Veamos ahora un ejemplo de definición de evento de tipo teclado:

```
//CODIGO HTML DE REFERENCIA
<input id = "nombre" type="text">
<input id = "edad"   type="number">
//CODIGO JS
let input1  = document.getElementById("nombre");
let input2  = document.getElementById("edad");
input1.onkeyup   = () => {console.log("keyUp")};
input2.onkeydown = () => {console.log("keyDown")};
```

Podemos notar en los ejemplos de eventos de mouse y teclado que empleamos las propiedades on-event para definir el manejador de eventos. Optamos por funciones flechas, y una salida por consola en respuesta, porque nos permite hacer un código más acotado, con la finalidad de exponer un ejemplo sencillo. No obstante, ante el desarrollo de la propia aplicación, se espera que en los manejadores de eventos se realice un procesamiento significativo, transformando las entradas mediante operaciones y efectuando al menos una salida por el DOM en respuesta.

Evento change

El evento change se activa ante un cambio de valor en elementos de tipo <input>, <select>, y <textarea>. Su principal utilidad es poder determinar cuándo el nuevo valor es confirmado por el usuario. Por ejemplo, mientras estamos escribiendo en un input de tipo texto, no hay evento change, ya que no se dispara por cada caracter ingresado en la caja; pero cuando salimos del input modificado, pasando a otro input o sección de la aplicación, el nuevo valor se asume como confirmado, disparando entonces el evento

change.

A continuación podemos observar cómo asignar el manejador de eventos empleando la propiedad ***onchange***:

```
//CODIGO HTML DE REFERENCIA
<input id = "nombre" type="text">
<input id = "edad"   type="number">
//CODIGO JS
let input1  = document.getElementById("nombre");
let input2  = document.getElementById("edad");
input1.onchange = () => {console.log("valor1")};
input2.onchange = () => {console.log("valor2")};
```

Evento submit

El evento submit se activa cuando un elemento de tipo formulario (<form></form>) es enviado. Este evento normalmente se utiliza para validar el formulario antes de ser enviado al servidor, o bien para abortar el envío y procesarlo con JavaScript de forma asíncrona. Veamos ahora un ejemplo codificado:

```
//CODIGO HTML DE REFERENCIA
 <form id="formulario">
      <input type="text">
      <input type="number">
      <input type="submit" value="Enviar">
 </form>
//CODIGO JS
let miFormulario      = document.getElementById("formulario");
miFormulario.addEventListener("submit", validarFormulario);

function validarFormulario(e){
    e.preventDefault();
```

```
    console.log("Formulario Enviado");
}
```

Podemos observar en el manejador de eventos una línea adicional; al recibir el parámetro cuyo identificador es “e”, llamamos en la primera línea de la función al método preventDefault. ¿Por qué? Esto se debe a que el método nos permite cancelar el comportamiento por defecto en respuesta al evento submit, el cual incluye refrescar la página actual. Como estamos programando aplicaciones interactivas, y pretendemos mostrar salidas al usuario sin que se refresque la página, es necesario emplear las mencionadas instrucciones.

Tenemos que identificar que refrescar la página actual es el equivalente en aplicaciones web a “reiniciar la aplicación”, y si no se establecen mecanismos de guardado previo (por ejemplo usando ***storage***), la información producida por el usuario antes del envío del formulario se pierde. Por esta razón, particularmente ante el evento ***submit***, es necesario emplear el ***preventDefault***; pero ¿de dónde salió el parámetro e? Esto lo veremos en la siguiente sección.

Información del evento

Vimos con el manejador del evento ***submit*** que en algunos casos es necesario obtener información del evento para poder realizar acciones, como prevenir el comportamiento por defecto para operar correctamente. Para esta situación, existe en JavaScript el objeto event.

En todos los navegadores modernos se crea de forma automática un parámetro único, que se pasa a la función manejadora referenciado al objeto event. Dicho parámetro puede usarse o no en la respuesta al evento, pero siempre estará disponible como

argumento.

En el siguiente ejemplo, además de emplear el método ***preventDefault*** para controlar la respuesta submit, utilizamos la propiedad ***target***, la cual permite obtener una referencia del elemento desde el cual ocurrió el evento, es decir, el propio formulario:

```
//CODIGO HTML DE REFERENCIA
 <form id="formulario">
      <input type="text">
      <input type="number">
      <input type="submit" value="Enviar">
    </form>
//CODIGO JS
let miFormulario      = document.getElementById("formulario");
miFormulario.addEventListener("submit", validarFormulario);

function validarFormulario(e){
    //Cancelamos el comportamiento del evento
    e.preventDefault();
    //Obtenemos el elemento desde el cual se disparó el evento
    let formulario = e.target
    //Obtengo el valor del primero hijo <input type="text">
    console.log(formulario.children[0].value);
    //Obtengo el valor del segundo hijo <input type="number">
    console.log(formulario.children[1].value);
}
```

Usar ***e.target*** es una forma de obtener el elemento del DOM al cual se asoció el evento, evitando la llamada mediante document.getElementById("formulario") para obtener el valor de sus inputs hijo.