M1 - 9 - JavaScript Eventos

Eventos

Los eventos son acciones o acontecimientos que ocurren dentro del navegador y que el navegador nos facilita por si queremos utilizarlos.
Por ejemplo, si el usuario hace clic en un botón en una página web, es posible que desee responder a esa acción mostrando un cuadro de información.

Hay muchos tipos diferentes de eventos que pueden ocurrir, por ejemplo:

- El usuario hace clic con el mouse sobre un elemento determinado o coloca el cursor sobre un elemento determinado.
- El usuario presiona una tecla en el teclado.
- El usuario cambia el tamaño o cierra la ventana del navegador.
- Una página web termina de cargar.
- Un formulario se envía
- Un video se reproduce, pausa o finaliza la reproducción.
- Un error ocurre.
- etc. Hay muchísimos eventos más que puedes consultar.

Objeto Event

Como decíamos antes, este tipo de acciones hacen que el navegador nos informe de ellas a través de un objeto conocido como **El Objeto Event.** Este objeto contiene toda la información acerca del evento que acaba de producirse, tiempo, contenido, tipo, métodos de control, listeners asociados, etc. Aquí podemos ver algo de la información que contiene:

Tipo	Nombre	Descripción
Propiedades de control del evento	type	Devuelve el tipo de evento producido, sin el prefijo on (p.ej. click)
	target	Devuelve el elemento del DOM que disparó el evento (inicialmente)
	currentTarget	Devuelve el elemento del DOM que está disparando el evento actualmente (no nesariamente el elemento que disparó el evento, ya que puede ser un disparo debido a burbujeo)
Otras propiedades de control del evento	eventPhase (indica en qué fase de tratamiento de evento estamos, 1 captura, 2 en objetivo, 3 burbujeo), bubbles (booleana, indica si es un evento que burbujea o no), cancelable (booleana, devuelve si el evento viene seguido de una acción predeterminada que puede ser cancelada), cancelBubble (booleana, devuelve si el evento actual se propagará hacia arriba en la jerarquía del DOM o no).	
Propiedad temporal	timeStamp	Devuelve una medida de tiempo en milisegundos desde un origen temporal determinado.
Propiedades de localización del puntero del ratón	clientX, clientY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda de la ventana del navegador y se expresan en pixeles.
	screenX, screenY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda de la pantalla y se expresan en pixeles.
	pageX, pageY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda del documento, que pueden ser distintas a las de la ventana si el usuario ha hecho scroll sobre el documento.

Propiedad para detectar el botón del ratón pulsado	button	Normalmente empleado para el evento mouseup (liberación de botón del ratón) para detectar cuál ha sido el botón pulsado. Contiene un valor numérico: 0 para click normal (botón izquierdo), 1 para botón central (botón en el scroll), 2 para botón auxiliar (botón derecho).
Propiedades relacionadas con el teclado	Para determinar qué tecla ha sido pulsada	Lo estudiaremos por separado en la siguiente entrega del curso
Propiedades relacionadas con drag and drop	Algunas no estandarizadas	dataTransfer, dropEffect, effectAllowed, files, types
Otras propiedades varias	Otras (algunas no estandarizadas)	x, y, layerx, layery, offsetX, offsetY, wheelDelta, detail, relatedNode, relatedTarget, view, attrChange, attrName, newValue, prevValue, data, lastEventId, origin, source
Método	stopPropagation()	Detiene la propagación del evento
Método	preventDefault()	Cancela (si es posible) la acción de defecto que debería ocurrir después del evento (equivale a return false para cancelar la acción).
Otros métodos	initEvent(a, b, c) se usa para definir eventos. No lo estudiaremos.	

Listener

Los listernes son “alarmas” que utilizamos para que en caso de producirse un evento concreto, se desencadene una acción que nosotros hayamos definido.

Hay muchos tipos de listeners y se pueden utilizar de muchas maneras diferentes pero para entender mejor el concepto, vamos a ver un ejemplo:

Podemos utilizar uno de los listener nativos en cualquier elemento de nuestra web, como por ejemplo,un listener **onclick** en un botón. Este llamará a una función (**controlador de eventos**) definida por nosotros `nextPage()`

```
<button onclick="nextPage()">+</button>
```

```
let page = 1;

function nextPage() {
  page = page + 1;
}
```

Esta función ejecutará un código concreto que nosotros habremos definido y que se ejecutará, por tanto, a demanda del listener, es decir, cuando ocurra el evento. (El usuario pincha en el botón y ejecuta el código).

También podemos agregar nosotros un listener a un elemento ya creado en nuestro DOM o a uno que hayamos creado mediante el método

```
.addEventListener('listener', callback)
```

A través del selector `.getElementById()` seleccionamos el elemento al que queremos ponerle el listener.

```
document.getElementById("titulo").addEventListener('click' , () =>
{console.log("elemento clickado!")})
```

De esta forma habremos acoplado el listener “click” al elemento cuyo id es “titulo” en nuestro DOM y le hemos indicado, mediante una función flecha, que cuando se dispare el mismo, nos imprima por la consola la frase “elemento clickado”.

Event Bubbling

El fenomeno “bubbling” y el “capturing” ocurre cuando esos eventos se propagan.
Por ejemplo aquí que tenemos un botón dentro de otro botón y dentro de un div en nuestro DOM:

```
<div id="mi_div">
  <button id="mi_boton_1">
    Boton 1
    <button id="mi_boton_2">Boton 2</button>
  </button>
</div>
```

Si le añadimos un listener a cada uno de estos elementos:

```
document.getElementById('mi_boton_1').addEventListener('click', () =>
console.log('boton 1 clickado!'))

document.getElementById('mi_boton_2').addEventListener('click', () =>
console.log('boton 2 clickado!'))

document.getElementById('mi_div').addEventListener('click', () =>
console.log('div clickado!'))
```

Ocurrirá que cuando apretemos el boton 2, este generará un evento, al mismo tiempo, generaremos un evento del boton 1 y luego otro del div que los contiene a todos, es decir, “burbujea” de abajo hacia arriba en nuestro DOM.

Por contra, si nos ocurre de arriba hacia abajo estaríamos hablando de “capturing” aunque esto no ocurre naturalmente, sino que hay que indicarle al listener que queremos capturar los eventos de los elementos del DOM descendientes, pasándole como tercer argumento al método, un objeto de configuración:

```
document.getElementById('mi_div').addEventListener('click', () =>
console.log('div clickado!!'), { capture: true })
```

Para prevenir el fenómeno del bubbling, utilizamos uno de los métodos que contiene el objeto Event natural del propio evento, el `.stopPropagation()` que nos llega de forma automática como argumento del callback del método `.addEventListener()`

```
document.getElementById('mi_boton_1').addEventListener('click', (e) => {
  e.stopPropagation()
  console.log('boton 1 clicado!')
})
```