EVENTOS

Un evento en JS es una acción u ocurrencia que sucede en un documento HTML, como hacer clic en un botón, mover el ratón, presionar una tecla o cargar una página. Estos eventos permiten que nuestras páginas web sean interactivas y respondan a las acciones del usuario.

Existen varios tipos de eventos:

- Eventos del Mouse
- ***click***: cuando se hace clic en un elemento (los dispositivos con pantalla táctil lo generan con un toque).
- ***contextmenu***: cuando se hace clic derecho en un elemento.
- ***mouseover*** / ***mouseout***: cuando el ratón está sobre un elemento o sale de él.
- ***mousedown*** / ***mouseup***: cuando se presiona / suelta el botón del ratón sobre un elemento.
- ***mousemove***: cuando se mueve el ratón.

- Eventos de Teclado
- ***keydown***: cuando se presiona una tecla.
- ***keyup***: cuando se suelta una tecla.
- ***keypress***: mientras se mantiene una tecla presionada.

- Eventos de Elementos de Formulario
- ***submit***: cuando el usuario envía un formulario `<form>`.
- ***reset***: cuando se borra el contenido de un formulario.
- ***focus***: cuando el visitante se centra en un elemento, por ejemplo, en un `<input>`.
- ***change***: cuando cambia el valor de algún elemento del formulario.

- Eventos del DOM:
- ***load***: cuando se carga y procesa el HTML; el DOM está completamente construido.
- ***resize***: cuando cambia el tamaño de la ventana del navegador.
- ***scroll***: cuando se desplaza la página.

Controladores de eventos

Para reaccionar ante los eventos, podemos asignar un controlador, una función que se ejecuta cuando sucede un evento.

Hay varias formas:

- Atributo **HTML** (no conveniente, utilizando `onclick`):

```
<!-- FORMA 1 -->
<button class="btn btn-primary" onclick="alert('Hola...')">Saludar</button>

<!-- FORMA 2 -->
<button class="btn btn-primary" onclick="saludar()">Saludar</button>

<script>
function saludar(){
alert ("Hola...")
 }
</script>
```

- Propiedades **DOM** (utilizando `on<event>`):

```
<body>
        <button class="btn btn-primary" id="miBoton">Saludar</button>
</body>

<script>
let miBoton=document.getElementById("miBoton")

// FORMA 1
// ********

/* miBoton.onclick=saludar()*/ // ESTO NO FUNCIONARIA PORQUE LE ASIGNAMOS LA EJECUCIÓN DE LA FUNCIÓN
miBoton.onclick=saludar // ESTO SI FUNCIONA PORQUE ASIGNA LA FUNCIÓN

function saludar(){
        alert ("Hola...")
}

// FORMA 2 - CON FUNCIÓN ANÓNIMA
// ******************************
miBoton.onclick=function(){
        alert ("Hola...")
}

// FORMA 2 - CON SINTAXIS fatArrow
// ******************************
miBoton2.onclick=()=>alert ("Hola...")
</script>
```

- **Listener de eventos** *(recomendado usar)*: El problema fundamental de las formas que hemos visto de asignar controladores es que no podemos asignar varios controladores a un evento. Si queremos ejecutar dos acciones diferentes con funciones diferentes no podemos asignar los dos controladores al mismo elemento, ya que el segundo reemplazara al primero. Nos gustaría asignar dos controladores de eventos para eso. Pero una nueva propiedad DOM sobrescribirá la existente. Para solucionar esto tenemos `addEventListener` y `removeEventListener`.
  - **Agregar un Listener de Evento:** Usando `addEventListener`, puedes agregar varios manejadores a un mismo evento sin que se sobrescriban.

```
// Definimos las funciones que queremos ejecutar cuando ocurra el evento
function handler1() {
    alert('Hola primera vez...');
}

function handler2() {
    alert('Hola segunda vez...');
}

// Seleccionamos el elemento
let miBoton = document.getElementById("miBoton");

// Asignamos ambos manejadores al mismo evento "click"
miBoton.addEventListener("click", handler1);
miBoton.addEventListener("click", handler2);
```

  - **Eliminar un Listener de Evento:** Para poder eliminar un listener, es fundamental que la función manejadora (el *handler*) esté almacenada en una variable o declarada como función nombrada. Esto es necesario porque `removeEventListener` requiere exactamente la misma referencia de función que `addEventListener`. No es posible eliminar un listener si se pasó una función anónima, ya que sería una instancia diferente.

```
// Definimos la función del controlador
function handler() {
    alert('Hola...');
}

// Seleccionamos el elemento
let miBoton = document.getElementById("miBoton");

// Agregamos el listener con el manejador de evento
miBoton.addEventListener("click", handler);
```

```
// Eliminamos el listener usando el mismo manejador
miBoton.removeEventListener("click", handler);
```

Propagación de eventos: Bubbling and Capturing

Cuando se hace click en un elemento dentro de una página web (por ejemplo, en un `<p>` que está dentro de un `<div>`), el navegador debe decidir en qué orden manejar esos eventos de clic entre los elementos involucrados (el `<p>` y el `<div>` en este caso). Existen dos formas de definir este orden:

- **Bubbling** (burbujeo): El evento se maneja primero en el elemento más profundo o interno (el hijo), y luego sube "haciendo burbuja" hacia los elementos contenedores. Si `useCapture` está en `false`, el evento burbujea.
- **Capturing** (captura): El evento se maneja primero en el elemento más externo (el contenedor o padre), y luego pasa al elemento más interno (hijo). Si `useCapture` está en `true`, el evento captura.

```
<div id="myDiv">
    <p>Este es un div. Haz clic en el botón de abajo:</p>
    <button id="myButton">¡Haz clic aquí!</button>
</div>

<script>
// Funciones para manejar los eventos de clic
function clickOnDiv() {
    alert("Click en el DIV de fondo coral");
}

function clickOnButton() {
        alert("Click en el BOTÓN de fondo blanco");
}

// Añadimos los eventos para el div y el botón con bubbling y capturing

// ********************************   CASO 1
*********************************
// Bubbling (burbujeo): de adentro hacia afuera
// Primero muestra clickOnButton(), luego clickOnDiv()
document.getElementById("myButton").addEventListener("click", clickOnButton,
false);
document.getElementById("myDiv").addEventListener("click", clickOnDiv, false);
```

```
// ********************************   CASO 2
**********************************
// Capturing (captura): de afuera hacia adentro
// Primero muestra clickOnDiv(), luego clickOnButton()
document.getElementById("myButton").addEventListener("click", clickOnButton,
true);
document.getElementById("myDiv").addEventListener("click", clickOnDiv, true);
</script>
```

Objeto event

El objeto `Event` se utiliza para manejar eventos (como clics de ratón o pulsaciones de teclado) de forma más detallada. Este objeto contiene información adicional sobre el evento que se activó, permitiendo, por ejemplo, conocer las coordenadas del ratón o la tecla específica que se presionó.

- Ejemplo básico: Al producirse un evento, el navegador crea un objeto `Event` que contiene todos los detalles del evento. Este objeto se pasa automáticamente al controlador que maneja el evento, lo que permite acceder a propiedades como:
  - `event.type`: indica el tipo de evento, como `"click"`.
  - `event.currentTarget`: hace referencia al elemento que disparó el evento.
  - `event.clientX` y `event.clientY`: representan las coordenadas del cursor en la ventana al momento del evento.
- Ejemplo en código: En este ejemplo, al hacer clic en un botón, se muestra una alerta con el tipo de evento y las coordenadas del cursor:

```
<button id="miBoton">Haz click...</button>
<script>
let miBoton = document.getElementById("miBoton");
miBoton.onclick = function(event) {
    alert("Evento " + event.type + " en " + event.currentTarget);
    alert("Coordenadas: " + event.clientX + ":" + event.clientY);
};
</script>
```

Objeto onload

Para acceder a los elementos del **DOM** (Document Object Model) en una página web, esta debe cargarse completamente, es decir, todos los elementos deben estar listos. Si intentamos usar JavaScript en el DOM antes de que la página termine de cargar, obtendremos errores porque los elementos aún no existen.

Existen varias formas de asegurarse de que el código JavaScript acceda al DOM solo cuando la página ya está lista:

1. **Ubicación de los Scripts**: Colocar los scripts de JavaScript al final del `<body>`, para que se ejecuten una vez que el contenido de la página esté cargado.
2. **Uso del evento** `onload`: Este evento se activa cuando la página está completamente cargada.
   - **Con el atributo HTML**: En la etiqueta `<body>`, podemos usar `onload` para llamar a una función cuando la página esté lista.
   - **Con el objeto** `window`: También podemos usar `window.onload` para asegurarnos de que toda la página (incluido el `body`) esté cargada.

     Ejemplos:
     - Llamar a una función existente con `window.onload = nombreFuncion;`
     - Usar una función anónima: `window.onload = function() { /* código */ }`
     - Usar una función "arrow" (fat arrow): `window.onload = () => { /* código */ }`
3. **addEventListener**: Otra opción es usar `addEventListener("load", ...)` sobre `window` para ejecutar el código una vez que la página esté cargada.
   - Funciona con una función existente, función anónima, o con "fat arrow".

Eventos personalizados

Los eventos personalizados son creados por los desarrolladores para representar acciones o estados específicos en una aplicación, permitiendo comunicación entre diferentes partes del código de manera flexible y desacoplada. Esto facilita la creación de flujos de trabajo específicos y la simplificación del código.

- Beneficios de los eventos personalizados
- **Comunicación entre componentes**: Facilitan la comunicación entre distintas partes de una aplicación sin necesidad de conexiones directas.
- **Flujos de trabajo personalizados**: Permiten crear eventos específicos para situaciones no cubiertas por los eventos nativos.
- **Código más claro**: Encapsulan la lógica de interacción, mejorando la legibilidad y el mantenimiento.

Creación y lanzamiento de un evento personalizado

Para crear un evento personalizado, se usa el constructor `CustomEvent`, que recibe:

1. **Nombre del evento**: Identifica el evento.

2. **Opciones**: Objeto con propiedades como `bubbles` (si el evento se propaga), `cancelable` (si puede cancelarse) y `detail` (datos adicionales).

Ejemplo:

```
const myEvent = new CustomEvent('myCustomEvent', {
    detail: {
        message: 'Hola desde el evento personalizado',
        data: 42
    }
});
```

Para lanzarlo, usamos `dispatchEvent()` sobre un elemento (en el ejemplo, sobre `window`):

```
window.dispatchEvent(myEvent);
```

Escuchar el evento personalizado

Para capturar el evento, se usa `addEventListener()` de forma similar a los eventos nativos. En el ejemplo, se accede a `event.detail` para obtener los datos:

```
window.addEventListener('myCustomEvent', (event) => {
    console.log(event.detail.message); // "Hola desde el evento personalizado"
});
```

Ejemplo completo

Un botón lanza el evento personalizado al hacer clic, y un listener captura y maneja el evento:

```
<body>
    <button id="miBoton">Lanzar evento</button>
</body>
<script>
    const myEvent = new CustomEvent('myCustomEvent', {
        detail: { message: 'Hola desde el evento personalizado', data: 42 }
    }); // crear evento

    let miBoton = document.getElementById("miBoton"); // asocia a elemto HTML
una variable

    miBoton.addEventListener("click", () => window.dispatchEvent(myEvent)); //
lanzamos evento
    window.addEventListener('myCustomEvent', (event) => {
```

```
        console.log(event.detail.message); // "Hola desde el evento
personalizado"
    });
</script>
```