TEMA 3. EVENTOS (I)

Un evento es una acción que ocurre sobre algún elemento HTML, como hacer clic en un botón, mover el mouse sobre una imagen, presionar una tecla, etc. Existen muchos tipos de eventos en HTML y JavaScript, como eventos de ratón (clic, desplazamiento, etc.), eventos de teclado, eventos de formulario (envío de formularios), eventos de carga de página, eventos de temporizador y muchos más.

Aquí, resulta claves “escuchar eventos”, es decir, se puede escuchar eventos que están asociados a elementos HTML para responder a ellos, esto se hace generalmente con JavaScript:

```
const miBoton = document.getElementById('miBoton');
miBoton.addEventListener('click', function() {
    // Este código se ejecutará cuando se haga clic en el botón.
});
```

En el ejemplo anterior, se utiliza una función llamada “manejadora de eventos”, que (tal y como cabe esperar) se ejecutan cuando ocurre el evento.

El método `**addEventListener**` en JavaScript se utiliza para adjuntar un escuchador de eventos a un elemento HTML, lo que permite que el elemento responda a ciertos tipos de eventos, como clics del ratón, pulsaciones de teclas, cambios en los campos de formulario, entre otros:

Sintaxis:

elemento.addEventListener(evento, función, useCapture);

- **elemento**: El elemento HTML al que deseas adjuntar el escuchador de eventos.
- **evento**: El tipo de evento que deseas escuchar.
- **función**: La función que se ejecutará cuando ocurra el evento.
- **useCapture** (opcional): Un valor booleano que indica si el evento debe ser capturado durante la fase de captura (true) o no (false).

La diferencia entre `function(event)` y `()` en los manejadores de eventos de JavaScript radica en cómo se maneja el objeto de evento y cuándo se pasa como argumento.

1. function(event): Cuando defines un manejador de eventos como una función con un argumento, llamado `event` o cualquier otro nombre, estás permitiendo que el objeto de evento se pase como un argumento a la función. Esto es útil cuando necesitas acceder a propiedades o información específica del evento dentro de la función. Por ejemplo:

```
elemento.addEventListener("click", function(event) {
  console.log("Se hizo clic en el elemento:", event.target);
  console.log("Coordenadas del clic (X, Y):", event.clientX, event.clientY);
});
```

En este caso, `event` es un objeto que contiene información sobre el evento de clic, y puedes acceder a sus propiedades, como `target` para obtener el elemento que desencadenó el evento, o `clientX` y `clientY` para obtener las coordenadas del clic.

```
miBoton.addEventListener('click', function(event) {
        console.log('Tipo de evento: ' + event.type);
        console.log('Elemento de destino: ' + event.target);
});
```

Propiedades del evento:

Los eventos en JavaScript vienen con una serie de propiedades y métodos que proporcionan información y control sobre el evento.

1. `**target**`: Hace referencia al elemento en el cual se originó el evento. Es útil para identificar cuál elemento desencadenó el evento.

EJERCICIO A: Muestra un mensaje que indica el elemento en el que se hizo clic dentro de un contenedor. Agrega tres botones dentro de un contenedor. Cuando se hace clic en uno de los botones, se dispara el evento de clic en el contenedor. Usamos la propiedad `target` para verificar si el elemento en el que se hizo clic es un botón, y luego mostramos un mensaje que indica en cuál de los botones hiciste clic según su ID.

2. `**type**`: Indica el tipo de evento que se produjo, como "click", "mouseover", "keydown", etc.

EJERCICIO B: Programa una web que muestre en un alert el tipo de evento que se produce al pulsar cualquiera de los 3 botones que tiene que tener el sitio. Para escuchar este tipo de eventos utiliza container.addEventListener("click", function(event) { …}

3. `**preventDefault**()`: Un método que permite prevenir el comportamiento predeterminado asociado al evento. Por ejemplo, evitar que un enlace navegue a una URL.

Prevenir comportamiento predeterminado:

- A menudo, puedes evitar que un evento realice su comportamiento predeterminado. Por ejemplo, puedes evitar que un enlace siga un hipervínculo, o un formulario se envíe, etc.

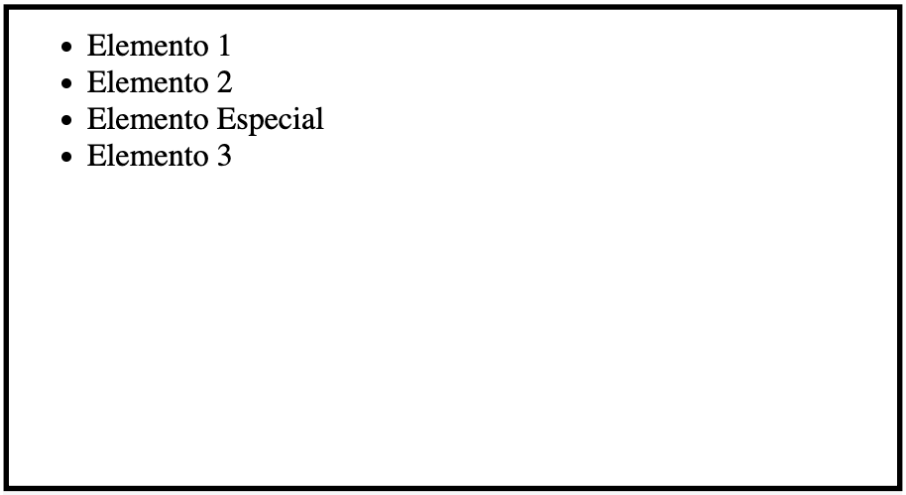
- Para hacer esto, puedes utilizar el método preventDefault() del objeto Evento en tu manejador de eventos.

```
miEnlace.addEventListener('click', function(event) {
        event.preventDefault(); // Evita el comportamiento predeterminado del enlace.
});
```

4. `**stopPropagation**()`: Un método que detiene la propagación del evento a través del modelo de eventos de burbujeo o captura. Útil para controlar cómo se manejan los eventos en elementos secundarios y padres.

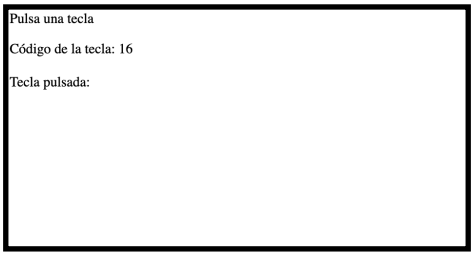
EJERCICIO C: Crea una página web que contenga una lista de elementos (<li>) y un párrafo vacío. Al hacer clic en cualquier elemento de la lista, se debe mostrar un mensaje en el párrafo que indique en cuál de los elementos de la lista se hizo clic, y al mismo tiempo, se mostrará un mensaje de alert indicando "Hiciste clic en la lista".

Sin embargo, en la lista, hay un elemento especial que tiene un comportamiento diferente. Al hacer clic en ese elemento, se debe mostrar un mensaje en el párrafo que indique "Hiciste clic en el elemento especial," pero no se debe mostrar el mensaje "Hiciste clic en la lista" en el alert.



5. `**keyCode**` y `**key**`: Propiedades asociadas a eventos de teclado que indican la tecla presionada y su código respectivamente.

EJERCICIO D: Programa una web que muestre por consola/alert/escribiendo en pantalla el código de la tecla y la tecla presionada.



6. `**clientX**` y `**clientY**`: Propiedades que proporcionan las coordenadas (en píxeles) del puntero del ratón en la ventana.

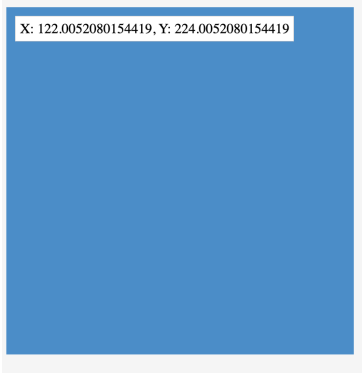
EJERCICIO E: En este caso, cuando se mueva del ratón sobre el contenedor, las coordenadas X e Y se muestran en la pantalla y el color de fondo del div cambia en función de la posición del cursor. Utiliza:

```
container.addEventListener("mousemove", function(event) {
        const x = event.clientX - container.getBoundingClientRect().left;
        const y = event.clientY - container.getBoundingClientRect().top;

        // Actualiza el texto con las coordenadas X e Y
        cursorCoordinates.textContent = `X: ${x}, Y: ${y}`;

        // Cambia el color de fondo basado en la posición del cursor
        const red = Math.floor((x / container.clientWidth) * 255); // clientWitdth devuelve el ancho interno del elemento
        const green = Math.floor((y / container.clientHeight) * 255);
        container.style.backgroundColor = `rgb(${red}, ${green}, 200)`;
    });
```

getBoundingClientRect() se utiliza para obtener la posición y el tamaño de un elemento en relación con la ventana gráfica o el área de visualización. En este caso, getBoundingClientRect().left se usa para obtener la distancia horizontal entre el borde izquierdo del elemento y el borde izquierdo de la ventana gráfica. Se resta esta distancia a la coordenada X absoluta del evento del ratón (event.clientX) para calcular la posición relativa del ratón dentro del elemento.



2. `()`: En algunos casos, puedes definir un manejador de eventos como una función sin argumentos (`()`). Esto se usa cuando no necesitas acceder directamente a las propiedades del objeto de evento o no necesitas información específica sobre el evento. Simplemente quieres que se ejecute una función cuando ocurra el evento:

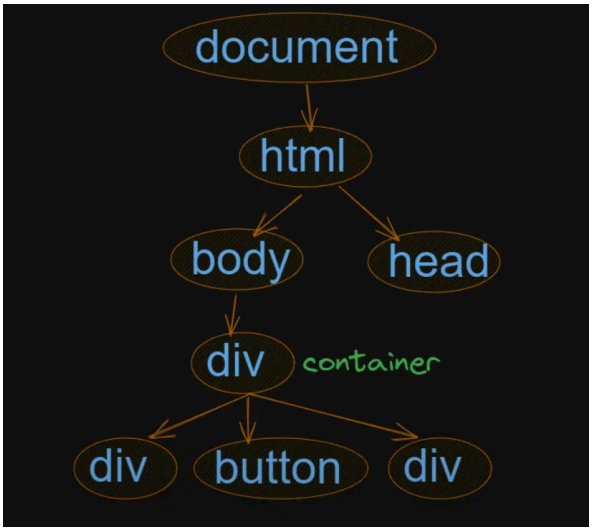
```
elemento.addEventListener("click", function() {
  console.log("Se hizo clic en el elemento.");
  // No se accede a event porque no se necesita información específica del evento.
});
```

Flujo de eventos:

1. Fase de captura
2. Objetivo
3. Burbujeo

La propagación de eventos es una forma de definir el orden de los elementos cuando ocurre un evento.

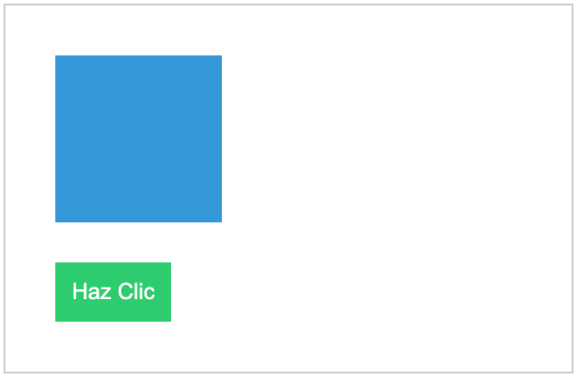
- Al **burbujear**, el evento del elemento más interno se maneja primero y luego el externo: primero se maneja el evento de clic del elemento <p> y luego el evento de clic del elemento - true. De arriba abajo.
- Al **capturar**, el evento del elemento más externo se maneja primero y luego el interno: el evento de clic del elemento se manejará primero, luego el evento de clic del elemento <p> - false. De **abajo arriba**



Ejercicio F: Escribe el código necesario para que al pulsar el botón este devuelva el mensaje "Botón pulsado", además al pulsar la caja azul (un div) también se debe devolver un mensaje "Caja azul pulsada", y finalmente, si se pulsa el contenedor blanco (un div), se muestra el mensaje "Caja blanca pulsada).

El contenedor blanco tiene dentro al azul y al botón.

La secuencia debe ser la siguiente: Botón pulsado, Caja Blanca pulsada.



Para evitar que se ejecute el mensaje de la caja blanca al pulsar el botón, utilizaríamos:

`if (event.target !== myButton) { ... }`:

En esta línea, estamos verificando si el objetivo del evento (el elemento que se hizo clic) no es igual al botón con el id "myButton".

```
container.addEventListener("click", function(event) {
        if (event.target !== myButton) {
            alert("¡Container Clickeado!");
        }
    });
```

removeEventListener

Es un método en JavaScript que se utiliza para eliminar un oyente de eventos previamente agregado a un elemento HTML. Esto es útil cuando ya no deseas que una función se ejecute en respuesta a un evento en particular o si deseas cambiar la función que se ejecuta en respuesta a ese evento.

La sintaxis general de `removeEventListener` es la siguiente:

elemento.removeEventListener(evento, función, useCapture);

- **elemento**: Es el elemento HTML al que se le ha agregado previamente un oyente de eventos.
- **evento**: Es una cadena que especifica el tipo de evento que se está escuchando. Por ejemplo, "click," "mouseover," "keydown," etc.
- **función**: Es la función que se ha asociado con el evento y que deseas eliminar. Debe ser la misma función que se utilizó al agregar el oyente de eventos.

EJERCICIO G: Programa el siguiente sitio web en el que se muestra en la parte inferior la posición un número aleatorio que se modifica siempre que el puntero del ratón se mueva por la zona naranja. Cuando se pulse el botón Remove el número no se modificará más.