Eventos

¿Qué es un event listener?

Un event listener en programación, especialmente en el contexto de JavaScript y la manipulación del Document Object Model (DOM), es una función que espera y reacciona a un evento en un elemento HTML. Los eventos pueden ser acciones del usuario como clics, pulsaciones de teclas, movimientos del ratón, o eventos del sistema como la carga de la página. Los event listeners son clave para la programación interactiva en la web.
Algunos aspectos clave:

- **Escuchar Eventos**: Un event listener "escucha" un tipo específico de evento en un elemento seleccionado.
- **Reacción a Eventos**: Cuando el evento especificado ocurre, el event listener ejecuta una función callback asociada.
- **Callback**: La función que se ejecuta en respuesta al evento. Puede ser una función anónima o una función nombrada. El callback puede depender de las propiedades del evento y su selector.
- **Uso en JavaScript**: En JavaScript, puedes usar el método addEventListener para agregar un escuchador a un elemento del DOM:

```
Unset
element.addEventListener('click', functionToCall);
element.addEventListener('click', () => {});
element.addEventListener('click', (event) => {});
```

  En estas líneas estás diciendo: "Cuando el usuario haga click en 'element', ejecuta functionToCall o la función anónima"
- **Remover Event Listeners**: También puedes remover un event listener si ya no es necesario, usando removeEventListener.

¿Cuáles son los eventos más utilizados?

A continuación te contamos cuáles son los eventos más importantes y cuando se ejecutan:

- **DOMContentLoaded**: Se dispara cuando el DOM ha sido completamente cargado y está listo para usar.
- **click**: Se dispara cuando un usuario hace clic en un elemento.
- **change**: Se dispara para elementos de entrada (input, select, textarea) cuando cambia su valor.
- **submit**: Se dispara cuando se envía un formulario.
- **dblclick**: Se activa cuando un usuario hace doble clic en un elemento.

¿Cómo se declaran los eventos?

Los eventos se pueden declarar en línea y con escuchadores.
Los eventos en línea se asignan directamente en el HTML. Esta sintaxis es muy útil aunque suele dificultar la mantenibilidad y legibilidad del código. La implementación de este tipo de escuchadores requiere:

- identificar la etiqueta a la cual se le asignará el escuchador de eventos
- agregar a la etiqueta el atributo **onTipo** para definir el tipo de evento a escuchar
- la función que ejecutará el evento

A modo de ejemplo:

```
Unset
<button id="boton" onclick="saludar()">Recibir saludo!</button>
//cada click sobre el botón ejecutará la función saludar

<input id="texto" type="text" onkeyup="filtrar(e)" />
//cada tecla presionada ejecutará la función filtrar
//en este caso la función depende de propiedades del evento “e”
```

Los escuchadores de eventos son otra forma de vincular un evento a un elemento HTML y lo hacen de manera programática, es decir, separando el HTML del JavaScript, lo cual es una práctica recomendada. Los escuchadores dependen de dos parámetros:

- tipo de evento
- función a ejecutar cuando suceda el evento

A modo de ejemplo:

```
Unset
const botonSelector = document.getElementById("boton");
botonSelector.addEventListener("click",saludar);
//cada click sobre el botón ejecutará la función saludar

const inputSelector = document.getElementById("texto");
inputSelector.addEventListener("keyup",(e)=>filtrar(e));

//cada tecla presionada ejecutará la función filtrar
//en este caso la función depende de propiedades del evento “e”
```

Presta atención al cambio de sintaxis: se debe pasar sólo la función (ejemplo saludar) no la ejecución de la función (ejemplo saludar())

¿Cuáles son las propiedades más importantes del evento?

Las propiedades más importantes del argumento **event** son:

- **target**: Representa el elemento que disparó el evento. Es un objeto con propiedades cómo (id, value, placeholder, type, del selector). Es muy útil cuando el mismo controlador de eventos se usa para múltiples elementos.
- **preventDefault**(): Es un método, no una propiedad. Llamándolo dentro de un controlador de eventos, puedes prevenir el comportamiento predeterminado del navegador para ese evento.
- **stopPropagation**(): Es otro método importante, que cuando se llama, evita que el evento se propague más allá del elemento actual.

Eventos I - Eventos de teclado

¿Cuáles son los eventos de teclado más utilizados?

Los eventos de teclado más utilizados en JavaScript son:

- **keydown**: Se dispara cuando una tecla es presionada.
- **keyup**: Ocurre cuando una tecla es liberada.
- **keypress**: Se lanza cuando una tecla presionada produce un carácter (obsoleto y no recomendado para uso nuevo).

A modo de ejemplo:

```
Unset
document.addEventListener("keydown", function(event) {
    console.log(`Tecla presionada: ${event.key}`);
});

document.addEventListener("keyup",(event)=> {
    console.log(`Tecla liberada: ${event.key}`);
});
```

Repaso I - Funciones de Orden Superior

¿Que es filter()?

El método filter es una función de orden superior en JavaScript que te permite filtrar los elementos de un arreglo basado en una condición definida en una función callback.
La función evalúa cada elemento del array, desde el primero hasta el último. El método filter retorna un nuevo arreglo que contiene exclusivamente sólo aquellos elementos que la condición de la callback retorna true.

A modo de ejemplo:

```
Unset
const numeros = [1, 12, 23, 8, 5, 7, 31];
const mayoresADiez = numeros.filter(cadaElemento => cadaElemento > 10);
console.log(mayoresADiez); // [12, 23, 31]
```

¿Para qué se utiliza filter()?

El método filter se utiliza para crear un nuevo array con todos los elementos que pasan la prueba implementada por la función proporcionada. La sintaxis básica del método filter es la siguiente:

```
Unset
array.filter((each, index, arr) => { })
```

- **array**: Es el array a iterar y evaluar con el método filter.
- **function**(currentValue, index, arr): Es la función de prueba para cada elemento del array y debe retornar una expresión que se resuelva a un valor booleano (true o false).
    - **each**: Es el valor del elemento actual que está siendo procesado en el array.
    - **index** (opcional): Es el índice del elemento actual en el array.
    - **arr** (opcional): Es el array al que pertenece el elemento actual.

El método **filter NO cambia el array original** sino que devuelve un nuevo array con los elementos que pasan la prueba. Si ningún elemento pasa la prueba, se devuelve un array vacío. En caso contrario si todos los elementos pasan la prueba, se devuelve el array completo.
Generalmente, el primer parámetro (función) es una función anónima de tipo flecha, pero puede ser una función definida.
Ejemplo 1:

```
Unset
const numbers = [1, 2, -3, 4, -5];
const positives = numbers.filter(number => number >= 0);
console.log(positives); // [1, 2, 4]
console.log(numbers); // [1, 2, -3, 4, -5]
```

Ejemplo 2:

```
Unset
const personas = [
    { nombre: "Ana", edad: 25, ciudad: "Madrid" },
    { nombre: "Juan", edad: 16, ciudad: "Barcelona" },
    { nombre: "Luis", edad: 17, ciudad: "Madrid" },
```

```
    { nombre: "Sofía", edad: 17, ciudad: "Valencia" },
    { nombre: "Carlos", edad: 22, ciudad: "Madrid" }
];

const mayoresMadrid = personas.filter(persona => {
    const esMayorDeEdad = persona.edad > 18;
    const esDeMadrid = persona.ciudad === "Madrid";
    // Retornar true si cumple ambas condiciones
    return esMayorDeEdad && esDeMadrid;
});
console.log(mayoresMadrid);

/* [
  { nombre: "Ana", edad: 25, ciudad: "Madrid" },
  { nombre: "Carlos", edad: 22, ciudad: "Madrid" }
] */
```