Eventos

¿Qué son los eventos?

Los eventos son acciones o sucesos que ocurren en el navegador que controlas o a los que responden mediante JavaScript. Hay varios tipos de eventos, como clicks del mouse, presiones de teclas, cambios en un formulario, etc. Cada evento puede ser capturado por un escuchador de eventos (event listener) para realizar una acción.

¿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 mientras que los escuchadores de eventos vinculan un evento a un elemento HTML de forma programática, es decir, separando el HTML del JavaScript, lo cual es una práctica recomendada.
A modo de ejemplo:

```
Unset
<button id="boton" onclick="saludar()">Recibir saludo!</button>
//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!

Eventos II - Eventos de mouse y change

¿Cuál es el evento de mouse más utilizado?

Los eventos de mouse son una parte fundamental de la interacción en aplicaciones web, permitiendo a los desarrolladores capturar y responder a diversas acciones realizadas con el mouse por parte del usuario. El evento de mouse más utilizado es **click** que se dispara cuando el usuario hace clic sobre un elemento, por ejemplo, para iniciar acciones como

abrir menús, enviar formularios, etc.

```
Unset
<button onclick="alert('Has hecho clic en el botón')">Haz clic aquí</button>

document.getElementById("miBoton").addEventListener("click", function() {
    alert("Has hecho clic en el botón");
});
```

¿Qué es un evento de cambio?

El evento onchange en JavaScript es una pieza fundamental en la interacción del usuario con formularios web y otros elementos interactivos. Este evento se dispara cuando el valor de un elemento <input>, <textarea> o <select> cambia y el elemento pierde el foco. Es decir, no se activa con cada golpe de tecla, sino cuando el usuario completa la entrada de datos en un campo y pasa a otro elemento de la página, o cierra el formulario, lo que marca el final de la edición de ese campo específico.
A modo de ejemplo:

```
Unset
<input type="text" name="nombre" onchange="alert('¡Cambió el texto!')">

document.getElementById("miElemento").addEventListener("change", ()=> {
    alert("¡El valor ha cambiado!");
});
```

Repaso II - Funciones de Orden Superior

¿Qué es map()?

El método map es una función de orden superior en JavaScript que te permite transformar los elementos de un arreglo en otro arreglo según las condiciones que se le pasen en una función callback.
La función evalúa cada elemento del array, desde el primero hasta el último. El método map retorna un nuevo arreglo que contiene exactamente la misma cantidad de elementos pero transformados.
A modo de ejemplo:

```
Unset
const numeros = [1, 12, 23, 8, 5, 7, 31];
const doble = numeros.map(cadaElemento => cadaElemento * 2);
console.log(doble); // [2, 24, 46, 16, 10, 14, 62]
```

¿Para qué se utiliza map()?

El método map se utiliza para crear un nuevo array con los resultados de la llamada a una función proporcionada aplicada a cada elemento del array original. La sintaxis básica del método map es la siguiente:

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

- **array**: Es el array a iterar y evaluar con el método map.
- **function**(currentValue, index, arr): Es la función de prueba para cada elemento del array y debe retornar el valor “transformado”.
    - **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 **map NO cambia el array original** sino que devuelve un nuevo array con los resultados de aplicar la función a cada elemento.
Ejemplo 1:

```
Unset
const personas = [
        { nombre: "Ana", edad: 25 },
        { nombre: "Juan", edad: 16 },
        { nombre: "Luis", edad: 30 }
];
const nombres = personas.map(persona => persona.nombre)
console.log(nombres) // ["Ana", "Juan", "Luis"]
```

Ejemplo 2:

```
Unset
const personas = [ { nombre: "Ana", edad: 25 },
                   { nombre: "Juan", edad: 16 },
                   { nombre: "Luis", edad: 30 }
];
const personasConMayoríaDeEdad = personas.map(persona => {
        return { ...persona, esMayorDeEdad: persona.edad >= 18 };
});

console.log(personasConMayoríaDeEdad);
/* [ { nombre: "Ana", edad: 25, esMayorDeEdad: true },
    { nombre: "Juan", edad: 16, esMayorDeEdad: false },
    { nombre: "Luis", edad: 30, esMayorDeEdad: true }
] */
```

La sintaxis **...persona (spread operator)** dentro del map, sirve para copiar el objeto completo persona, para luego agregar un nuevo campo (En este caso esMayorDeEdad).