JS SEMINARIO DE LENGUAJES

EXPLICACIÓN PRÁCTICA 2 (PARTE 2)

EVENTOS EN JAVASCRIPT

Manejo de eventos, manipulación del DOM y timers

AGENDA

1. Introducción A Los Eventos
2. ¿Cómo Se Usan?
3. Events, Event Handlers Y Listeners
4. Programación De Eventos Con Js
5. Eventos Por Html Vs Javascript
6. Eventos Comunes Y Ejemplos
7. Timers

INTRODUCCIÓN A LOS EVENTOS

Son acciones que ocurren en el navegador: clicks, escritura, envío de formularios, etc.

Permiten que la página responda a la interacción del usuario.

Ejemplo: al hacer click en un botón, se puede mostrar un mensaje o cambiar el contenido de la página.

ALGUNOS TIPOS DE EVENTOS

- **Mouse:** click, doble click, movimiento, scroll.
- **Teclado:** presionar tecla, soltar tecla.
- **Formulario:** enviar, cambiar valor de un elemento.
- **Ventana:** cargar, redimensionar, cerrar.

Hay muchos mas: clipboard, touch, media, etc.

¿CÓMO SE USAN?

- **Desde HTML:** usando atributos como `onclick`.
- **Desde JS:** con `addEventListener` (más flexible y recomendado).

EJEMPLO BÁSICO: CLICK EN UN BOTÓN

```
<button onclick="alert('¡Hiciste click!')">Hacé clic</button>
```

Asociar la función desde HTML es directo, pero poco reutilizable.

EVENTS, EVENT HANDLERS Y LISTENERS

¿QUÉ ES UN EVENT HANDLER?

Un **event handler** (o *manejador de evento*) es una función que se ejecuta cuando ocurre un evento en un elemento.

JavaScript permite asociar estas funciones para responder a interacciones del usuario como clicks, escritura, movimientos del mouse, etc.

¿QUÉ HACE UN HANDLER? [1]

- Espera a que ocurra un evento específico (como un click).
- Cuando el evento ocurre, se ejecuta automáticamente.
- Recibe un parámetro llamado `event` con información del evento.

¿QUÉ HACE UN HANDLER? [2]

En este ejemplo, `mostrarAlerta` es el event handler del evento `click`.

```
<button onclick="mostrarAlerta()">Clickeame</button>

<script>
    function mostrarAlerta() {
        alert("¡Evento detectado!");
    }
</script>
```

¿QUÉ ES EL `event`?

Es un objeto especial que contiene información sobre lo que ocurrió.

```
function manejarClick(event) {
    console.log("Tipo:", event.type);
    console.log("Target:", event.target);
    console.log("Valor:", event.target.value);
}
```

`event.preventDefault()`

Evita el comportamiento por defecto del navegador (como enviar un formulario).

```
function enviarFormulario(event) {
    event.preventDefault();
    console.log("Formulario detenido");
});
```

PROGRAMACIÓN DE EVENTOS CON JS

Se pueden agregar eventos a elementos HTML usando JavaScript.

Esto permite una mayor flexibilidad y separación de responsabilidades.

Se pueden agregar múltiples eventos a un mismo elemento.

¿QUÉ ES UN LISTENER?

Es el vínculo entre un evento y su handler.

Escucha un evento específico y ejecuta el handler cuando ese evento ocurre.

AGREGAR EVENTOS

`addEventListener`: permite registrar eventos desde JS.

```
1 const btn = document.getElementById("btn");
2
3 function saludar() {
4     alert("¡Hola!");
5 }
6
7 btn.addEventListener("click", saludar);
```

REMOVER EVENTOS

`removeEventListener:` permite remover un evento (solo si es la misma función).

```
const btn = document.getElementById("btn");
btn.removeEventListener("click", saludar);
```

COMPARACIÓN

DESDE HTML

```html
<button onclick="alert('¡Hola!')">
    Clickeame
</button>
```

DESDE JS

```html
<button id="btn">Clickeame</button>

<script>
    const btn = document.getElementById("
    btn.addEventListener("click", () => {
        alert("¡Hola!");
    });
</script>
```

DELEGACIÓN DE EVENTOS [1]

Permite manejar eventos en elementos hijos desde un elemento padre.

Ejemplo: manejar clicks en una lista sin agregar un listener a cada elemento.

```
const lista = document.getElementById("lista");
lista.addEventListener("click", (event) => {
    if (event.target.tagName === "li") {
        alert("Hiciste click en: " + event.target.textContent);
    }
});
```

DELEGACIÓN DE EVENTOS [2]

La delegación de eventos mejora el rendimiento al evitar agregar múltiples listeners.

En lugar de agregar un listener a cada elemento hijo, se agrega uno solo al padre.

Esto reduce el uso de memoria y mejora la eficiencia.

Ejemplo: si tienes una lista de elementos, puedes agregar un solo listener al contenedor.

EVENTOS POR HTML VS JAVASCRIPT

Desde HTML	Desde JS
`onclick="..."`	`addEventListener`
Mala separación de responsabilidades	Separación limpia entre estructura y lógica
No reutilizable	Reutilizable y flexible

EVENTOS COMUNES Y EJEMPLOS

ALGUNOS EVENTOS COMUNES

- `click`: al hacer click
- `submit`: al enviar formulario
- `change`: al cambiar valor de un input
- `load`: al cargar la página
- `keydown`: al presionar tecla

VALIDACIÓN EN TIEMPO REAL

```
const input = document.getElementById("nombre");
const mensaje = document.getElementById("mensaje");

input.addEventListener("input", () => {
    mensaje.textContent = input.value.length < 3 ? "Muy corto" : "";
});
```

OBTENER VALOR DE UN SELECT

```
<select id="color">
    <option value="red">Rojo</option>
    <option value="blue">Azul</option>
</select>

<script>
    const select = document.getElementById("color");

    select.addEventListener("change", function () {
        const color = select.value;
        document.body.style.backgroundColor = color;
    });
</script>
```

TIMERS

Los **timers** nos permiten ejecutar código con cierto retardo o de forma repetida en el tiempo.

Se usan junto a eventos para crear dinámicas como cronómetros, slideshows o cambios automáticos al interactuar con el usuario.

TIPOS DE TIMERS

- `setTimeout(función, ms)`: ejecuta una vez luego de un retardo.
- `setInterval(función, ms)`: ejecuta repetidamente cada cierto tiempo.
- `clearInterval(id)`: detiene una repetición iniciada con `setInterval`.

EJEMPLO CON `setInterval`

```
let contador = 0;

const id = setInterval(() => {
    console.log("Contador:", ++contador);
    if (contador === 5) clearInterval(id); // corta luego de 5 veces
}, 1000);
```

⏳ EJEMPLO CON `setTimeout`

```
setTimeout(() => {
    console.log("¡Hola después de 2 segundos!");
}, 2000);
```

FIN DE LA CLASE