JS

listeners

Aquí tienes una versión mejorada del código que usa addEventListener en lugar de oninput en el HTML:

**Código HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width">

<title>JavaScript Number Input Event Listener</title>

<script defer src="action.js"></script>

</head>

<body>

<h1>Validación de <code>&lt;input type=number&gt;</code> con Event Listener</h1>

<p>Escriba un número o use las flechas.</p>

<label>

Escriba un número:

<input id="numberInput" type="number" min="1" max="12" step="0.1">

</label>

<p id="sliderValue">Valor seleccionado: </p>

</body>

</html>

**Código action.js:**

document.addEventListener("DOMContentLoaded", function() {

// Seleccionamos el input y el párrafo de salida

var inputField = document.getElementById("numberInput");

var output = document.getElementById("sliderValue");

// Agregamos el event listener para detectar cambios en el input

inputField.addEventListener("input", function(event) {

var val = event.target.value;

output.innerHTML = "Valor seleccionado: " + val;

});

});

**Explicación:**

1. Esperamos a que el DOM se cargue con DOMContentLoaded.
2. Seleccionamos el campo <input> y el elemento donde mostraremos el valor.
3. Usamos addEventListener("input", function(event) {...}) en lugar de oninput en el HTML.
4. Cada vez que el usuario escriba o cambie el valor, el texto en #sliderValue se actualizará dinámicamente.

📌 **Ventajas de esta versión:**

* Separa la lógica JavaScript del HTML.
* Mejora la legibilidad y el mantenimiento del código.
* Evita definir funciones globales innecesarias.

¿Necesitas agregar validaciones o mensajes adicionales? 😊

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width">

  <title>JavaScript slider use</title>

  <script>

function doSomething(evt) {

    // this is the slider value

    var val = evt.target.value;

    // get the output div

    var output = document.querySelector('#sliderValue');

    // display the value typed in the div

    output.innerHTML = "Value selected: " + val;

  }

</script>

</head>

<body>

  <h1>Simple <code>&lt;input type=range&gt;</code> field validation using the 'input' event</h1>

  <p>Just move the slider</p>

<label>

 1 <input type="range"

         min=1

         max=12

          step=0.1

         oninput = "doSomething(event)"> 12

</label>

  <p>

  <span id="sliderValue"></span>

</p>

</body>

</html>

La línea de código:

if (event.key === "Enter" && inputField.value.trim() !== "") {

se usa para asegurarnos de que solo se agregue un nuevo elemento <li> a la lista cuando el usuario presiona la tecla **"Enter"** y el input **no esté vacío**. Veamos el desglose:

**Explicación paso a paso:**

1. **event.key === "Enter"**
   * event.key obtiene la tecla que el usuario presionó.
   * "Enter" es el valor asociado a la tecla **Enter** en los eventos de teclado.
   * Esto significa que el código dentro del if solo se ejecutará cuando el usuario presione **Enter**.
2. **&& (Operador lógico AND)**
   * Se usa para asegurarnos de que ambas condiciones sean verdaderas al mismo tiempo.
   * Solo si el usuario presiona **Enter** **y** el input no está vacío, se ejecutará el código dentro del bloque { ... }.
3. **inputField.value.trim() !== ""**
   * inputField.value obtiene el texto que el usuario escribió en el input.
   * .trim() elimina los espacios en blanco al inicio y al final del texto.
   * !== "" verifica que el input **no esté vacío** después de eliminar espacios.
   * Esto evita que el usuario agregue elementos vacíos a la lista al presionar Enter con el input vacío.

**Ejemplo de cómo funciona**

| **Input del usuario** | **Tecla presionada** | **¿Se agrega un <li>?** | **Explicación** |
| --- | --- | --- | --- |
| "Hola" | Enter | ✅ Sí | Tiene texto y Enter fue presionado |
| " " | Enter | ❌ No | Se considera vacío después de .trim() |
| " Hola " | Enter | ✅ Sí | Se eliminan espacios y queda "Hola" |
| "Test" | Tecla "A" | ❌ No | No se presionó Enter |

**Conclusión**

Esta línea evita que se agreguen elementos vacíos a la lista y solo permite agregar un <li> cuando el usuario realmente quiere confirmar su entrada presionando **Enter**. 🚀

Aquí hay algunos problemas en tu código que explican por qué el botón de tipo submit no se está mostrando y cómo hacer que agregue un elemento <li> cuando se presione:

**Errores en el código:**

1. **El botón no aparece porque tiene un error tipográfico:**
2. inputSend.setTimeout('type', 'submit');
   * Debe ser inputSend.setAttribute("type", "submit");
3. **El botón no tiene un texto o valor visible:**
   * Se debe agregar un value para que sea visible, por ejemplo:
   * inputSend.setAttribute("value", "Agregar");
4. **La lógica de creación de <li> aún depende de keydown (Enter).**
   * En su lugar, debemos hacer que ocurra cuando el usuario presione el botón.

**Código corregido y mejorado:**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<style>

main {

display: flex;

flex-direction: column;

}

</style>

</head>

<body>

<main>

<div id="id1">

<p>

Vamos a crear un input dinámicamente. En tres segundos debe aparecer y

podrás usarlo. Escribe algo y presiona el botón "Agregar" para añadirlo a la lista.

</p>

</div>

<div id="id2">

<p id="display"></p>

</div>

<div id="id3"></div>

</main>

<script>

let display = document.querySelector("#display"); // Para mostrar mensajes

let listaCreada = false; // Controla si la lista ya fue creada

setTimeout(() => {

// Crear el input de texto dinámicamente

let inputField = document.createElement("input");

inputField.setAttribute("id", "enter");

inputField.setAttribute("type", "text");

document.querySelector("#id1").appendChild(inputField);

// Crear el botón tipo submit

let inputSend = document.createElement("input");

inputSend.setAttribute("id", "submit");

inputSend.setAttribute("type", "submit");

inputSend.setAttribute("value", "Agregar"); // Texto visible en el botón

document.querySelector("#id1").appendChild(inputSend);

// Evento para mostrar mensaje cuando el input obtiene foco

inputField.addEventListener("focus", () => {

display.innerHTML = "Atención, alguien va a escribir.";

});

// Evento para mostrar lo que se está escribiendo en tiempo real

inputField.addEventListener("input", () => {

display.innerHTML = inputField.value;

});

// Evento para agregar un elemento <li> cuando se presiona el botón

inputSend.addEventListener("click", () => {

if (inputField.value.trim() !== "") { // Evita agregar elementos vacíos

if (!listaCreada) {

let lista = document.createElement("ol");

lista.setAttribute("id", "lista");

document.querySelector("#id3").appendChild(lista);

listaCreada = true;

}

let lista = document.querySelector("#lista");

let itemLista = document.createElement("li");

itemLista.innerHTML = inputField.value;

lista.appendChild(itemLista);

inputField.value = ""; // Limpiar input después de agregar el elemento

}

});

}, 3000); // Se ejecutará después de 3 segundos

</script>

</body>

</html>

**Mejoras y cambios:**

✅ **El botón ahora aparece correctamente y tiene texto visible.**  
✅ **El evento de agregar <li> ahora ocurre cuando se presiona el botón "Agregar".**  
✅ **Se evita agregar elementos vacíos a la lista.**  
✅ **Se mantiene la limpieza del input después de agregar un elemento.**

🔹 **Ahora puedes escribir algo en el input y presionar el botón "Agregar" para añadirlo a la lista.** 🚀

Voy a explicarte el código en detalle y después te mostraré cómo modificarlo para que el mensaje cambie cuando se presione "Enter".

**Explicación del código original**

**HTML**

<main>

<div id="div1">

<h1>Simulacion de clck con el teclado</h1>

</div>

<div id="boton">

<button id="bt">Haz click o pulsa enter</button>

</div>

<div id="mensaje"></div>

</main>

* Hay un div que contiene un botón con el id="bt".
* Hay otro div vacío con el id="mensaje" donde se mostrará el mensaje cuando se haga clic en el botón o se presione "Enter".

**CSS**

main {

display: flex;

flex-direction: column;

gap: 2rem;

}

* Se usa display: flex con flex-direction: column para organizar los elementos en columna.
* gap: 2rem añade espacio entre los elementos.

**JavaScript**

const boton = document.querySelector('#boton');

const mensaje = document.querySelector('#mensaje');

* Se seleccionan los elementos boton y mensaje.

boton.addEventListener('click', () => {

mensaje.textContent = 'Has pulsado el botón';

});

* Se agrega un **event listener** para que, cuando el usuario haga clic en el botón, el mensaje en el div cambie a "Has pulsado el botón".

document.addEventListener("keydown", (event) => {

if(event.key === 'Enter'){

let clickEvent = new event('click');

boton.dispatchEvent(clickEvent);

}

});

* Se escucha el evento keydown (tecla presionada).
* Si la tecla presionada es "Enter", se crea un **evento de clic** y se envía (dispatchEvent) al botón.
* **Error en el código:** let clickEvent = new event('click'); debería ser let clickEvent = new Event('click'); con "E" mayúscula.

**Cómo modificarlo para que el mensaje cambie si se presiona Enter**

Cambiaremos el código para que el mensaje sea diferente si se presiona "Enter" en lugar de hacer clic en el botón.

**Código corregido y mejorado**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<style>

main {

display: flex;

flex-direction: column;

gap: 2rem;

}

</style>

</head>

<body>

<main>

<div id="div1">

<h1>Simulación de click con el teclado</h1>

</div>

<div>

<button id="bt">Haz click o pulsa Enter</button>

</div>

<div id="mensaje"></div>

</main>

<script>

const boton = document.querySelector('#bt');

const mensaje = document.querySelector('#mensaje');

// Evento de clic en el botón

boton.addEventListener('click', (event) => {

if (event.detail === 0) {

// Si detail es 0, significa que el evento no proviene de un clic real, sino de JavaScript

mensaje.textContent = 'Has pulsado Enter';

} else {

mensaje.textContent = 'Has pulsado el botón';

}

});

// Evento de teclado para detectar la tecla Enter

document.addEventListener("keydown", (event) => {

if (event.key === 'Enter') {

let clickEvent = new MouseEvent('click', { bubbles: true, cancelable: true });

boton.dispatchEvent(clickEvent);

}

});

</script>

</body>

</html>

**Explicación de las mejoras**

**Corrección de dispatchEvent**

Antes estaba escrito:

let clickEvent = new event('click');

Ahora se ha corregido:

let clickEvent = new MouseEvent('click', { bubbles: true, cancelable: true });

* MouseEvent('click') simula un clic real.
* Se agregan opciones como bubbles: true para que el evento se propague correctamente.

**Diferenciar entre clic real y tecla Enter**

Modificamos el addEventListener del botón:

boton.addEventListener('click', (event) => {

if (event.detail === 0) {

mensaje.textContent = 'Has pulsado Enter';

} else {

mensaje.textContent = 'Has pulsado el botón';

}

});

* event.detail indica el número de clics reales. Si es 0, significa que el evento se generó por código (es decir, con dispatchEvent y no con un clic real).
* Si el usuario hace clic con el mouse, el mensaje será **"Has pulsado el botón"**.
* Si el usuario presiona "Enter", el mensaje será **"Has pulsado Enter"**.

**Resumen de mejoras**

✅ **Corrige el error en dispatchEvent.**  
✅ **Diferencia entre clic real y tecla Enter.**  
✅ **Mantiene la misma funcionalidad con código más limpio.**

Si tienes dudas o quieres más mejoras, dime. 😊

**¿Qué hace dispatchEvent en JavaScript?**

El método **dispatchEvent(evento)** se usa para **disparar manualmente** un evento en un elemento del DOM. Esto significa que puedes simular eventos como clics, presionar teclas, o cualquier otro evento sin que el usuario tenga que hacer una acción física.

**Ejemplo básico de dispatchEvent**

const boton = document.querySelector("#miBoton");

// Crear un evento de clic manualmente

const eventoClick = new Event("click");

// Disparar el evento en el botón

boton.dispatchEvent(eventoClick);

📌 **Esto hace que el botón actúe como si hubiera sido clicado, incluso si el usuario no hizo clic.**

**¿Cómo funciona dispatchEvent en tu código?**

En tu código original, se usa dispatchEvent para simular un clic en el botón cuando el usuario presiona la tecla "Enter":

document.addEventListener("keydown", (event) => {

if (event.key === "Enter") {

let clickEvent = new Event("click");

boton.dispatchEvent(clickEvent);

}

});

📌 **Lo que sucede aquí:**

1. **Detecta que se ha presionado "Enter"** con keydown.
2. **Crea un evento de tipo "click"** con new Event("click").
3. **Dispara el evento en el botón** con boton.dispatchEvent(clickEvent).
4. **El botón recibe un evento de clic como si hubiera sido presionado con el mouse** y ejecuta su código normalmente.

**Tipos de eventos que se pueden disparar con dispatchEvent**

Puedes usar dispatchEvent con **cualquier tipo de evento**, pero debes crear el evento adecuado.

**Ejemplo con MouseEvent (clic con el mouse)**

const clickEvent = new MouseEvent("click", { bubbles: true, cancelable: true });

boton.dispatchEvent(clickEvent);

📌 Esto simula un clic real del mouse.

**Ejemplo con KeyboardEvent (presionar tecla)**

const enterEvent = new KeyboardEvent("keydown", { key: "Enter" });

document.dispatchEvent(enterEvent);

📌 Esto simula que el usuario ha presionado la tecla "Enter".

**¿Cuál es la diferencia entre dispatchEvent y llamar a una función directamente?**

Supongamos que tienes este código:

boton.addEventListener("click", () => {

console.log("Botón clicado");

});

**Opción 1: Ejecutar la función directamente**

console.log("Botón clicado"); // Se ejecuta la acción sin evento.

✅ **No genera eventos, solo ejecuta la acción.**

**Opción 2: Usar dispatchEvent**

boton.dispatchEvent(new Event("click"));

✅ **Simula que el botón fue clicado realmente, activando cualquier otro evento relacionado.**

📌 **Ventaja de dispatchEvent:** Si hay otros eventos (como mouseover, focus, etc.), estos también pueden ejecutarse automáticamente.

**Conclusión**

🔹 **dispatchEvent permite simular eventos como clics o pulsaciones de teclas sin la intervención del usuario.**  
🔹 **Es útil para automatización, testing o interacción avanzada en la web.**  
🔹 **Diferente de llamar directamente a una función porque realmente dispara eventos en el DOM.**

💡 **Ejemplo práctico:** Si tienes validaciones en un formulario y quieres probarlas automáticamente sin escribir en los campos, puedes usar dispatchEvent para simular la interacción. 🚀



La diferencia principal entre las dos peticiones radica en el tipo de solicitud HTTP que realizan:

1. **Primera petición (GET)**:
   * Se usa fetch('https://api.example.com/data') sin especificar un método, lo que por defecto significa que es una **solicitud GET**.
   * Se usa para **obtener datos** desde el servidor.
   * Se maneja el error si la respuesta no es exitosa (!response.ok).
   * Se convierte la respuesta a formato JSON con response.json() y se muestra en consola.
2. **Segunda petición (POST)**:
   * Usa fetch('https://api.example.com/submit', { method: 'POST', ... }), lo que indica que es una **solicitud POST**.
   * Se envían datos al servidor (body: JSON.stringify({ name: 'Juan', age: 25 })).
   * Incluye encabezados (headers) para indicar que el contenido es de tipo JSON.
   * Se usa para **enviar datos** al servidor en lugar de solo recuperarlos.

En resumen:

* **GET**: Se usa para obtener datos.
* **POST**: Se usa para enviar datos al servidor.

Ambas utilizan fetch(), promesas y .json() para manejar la respuesta.