Eventos y Manejo de Eventos con JavaScript

1. Introducción a los Eventos en JavaScript

- Definición de Eventos: Los eventos en JavaScript son acciones o sucesos que ocurren en el navegador, como el clic de un botón, el envío de un formulario, la carga de una página, entre otros.
- Importancia de los Eventos: Permiten interactuar con los usuarios y hacer que las aplicaciones sean dinámicas y reactivas. A través del manejo de eventos se puede responder a las acciones de los usuarios en tiempo real.

2. Tipos de Eventos Comunes en JavaScript

- Eventos de Mouse:
  - `click`: Se dispara cuando se hace clic en un elemento.
  - `dblclick`: Se dispara cuando se hace doble clic en un elemento.
  - `mouseover`: Ocurre cuando el puntero del ratón pasa sobre un elemento.
  - `mouseout`: Ocurre cuando el puntero del ratón sale de un elemento.
- Eventos de Teclado:
  - `keydown`: Se dispara cuando se presiona una tecla.
  - `keyup`: Se dispara cuando se libera una tecla.
  - `keypress`: Se dispara cuando se presiona una tecla, pero antes de que se libere.
- Eventos de Documento y Ventana:
  - `load`: Ocurre cuando la página se ha cargado completamente.
  - `resize`: Se dispara cuando cambia el tamaño de la ventana.
  - `scroll`: Ocurre cuando se desplaza la barra de desplazamiento de un elemento.
- Eventos de Formulario:
  - `submit`: Se dispara cuando se envía un formulario.
  - `focus`: Ocurre cuando un elemento obtiene el foco.
  - `blur`: Ocurre cuando un elemento pierde el foco.

3. Manejo de Eventos en JavaScript

- Agregar Manejadores de Eventos:
  - Atributo `onclick` en HTML: Se define directamente en el HTML.

```
<button onclick="alert('¡Hola!')">Haz clic aquí</button>
```

  - Propiedades de Eventos: Se asigna un manejador de eventos directamente en el script.

```
document.getElementById("miBoton").onclick = function() {
    alert('¡Botón clickeado!');
};
```

  - Método `addEventListener`: La forma más recomendada para agregar eventos, ya que permite añadir múltiples manejadores al mismo evento y quitar manejadores específicos.

```
document.getElementById("miBoton").addEventListener("click", function() {
    alert('Evento con addEventListener');
});
```

- Eliminar Manejadores de Eventos:
  - Se utiliza el método `removeEventListener` para eliminar un evento que se haya registrado con `addEventListener`.

```
function miFuncion() {
    alert('¡Evento eliminado!');
}
const boton = document.getElementById("miBoton");
boton.addEventListener("click", miFuncion);
boton.removeEventListener("click", miFuncion);
```

4. Propagación de Eventos

- Fase de Propagación:
  - Captura: El evento se propaga desde el elemento raíz hasta el objetivo.
  - Objetivo: El evento se dispara en el objetivo.
  - Burbuja: El evento se propaga de regreso desde el objetivo hasta el elemento raíz.
- Control de Propagación:
  - `stopPropagation()`: Detiene la propagación del evento hacia arriba o hacia abajo en el DOM.
  - `preventDefault()`: Cancela el comportamiento predeterminado del evento (por ejemplo, evitar que un enlace navegue a otra página).

```
document.getElementById("miEnlace").addEventListener("click", function(event) {
    event.preventDefault(); // Evita que el enlace siga el link
    alert('¡Enlace clickeado, pero no redirige!');
});
```

5. Delegación de Eventos

- Concepto: La delegación de eventos se refiere a asignar un único manejador de eventos a un contenedor común, en lugar de asignar múltiples manejadores a elementos secundarios.
- Beneficios:
  - Mejor rendimiento en aplicaciones con muchos elementos.
  - Manejo de eventos para elementos dinámicos.

- Ejemplo:

```
document.getElementById("contenedor").addEventListener("click", function(event) {
    if (event.target.tagName === "BUTTON") {
        alert('¡Botón en el contenedor clickeado!');
```

```
    }
});
```

6. Eventos Personalizados

- Crear y Despachar Eventos Personalizados:

```
// Crear un evento personalizado
let eventoPersonalizado = new CustomEvent("miEvento", {
    detail: { mensaje: "¡Hola, evento personalizado!" }
});

// Escuchar el evento
document.addEventListener("miEvento", function(e) {
    console.log(e.detail.mensaje); // Salida: ¡Hola, evento personalizado!
});

// Despachar el evento
document.dispatchEvent(eventoPersonalizado);
```

7. Aplicaciones Prácticas

- Formulario de Validación en Tiempo Real:

  Validar la entrada de datos de un formulario mientras el usuario escribe, mostrando mensajes de error en tiempo real.

- Interacciones Dinámicas con el DOM:

  Crear menús desplegables, galerías de imágenes interactivas y efectos de desplazamiento utilizando eventos.

- Juego Básico con Eventos:

  Desarrollar un pequeño juego (por ejemplo, "atrapar el cuadro") donde el usuario tenga que hacer clic en un cuadro que cambia de posición cada vez que se hace clic.

8. Buenas Prácticas en el Manejo de Eventos

- Utilizar `addEventListener` en lugar de propiedades de eventos.
- Evitar la anidación innecesaria de eventos para mejorar el rendimiento.
- Usar `removeEventListener` cuando un evento ya no sea necesario.
- Aplicar la delegación de eventos cuando sea posible para reducir el número de manejadores asignados.

9. Recursos Adicionales y Ejercicios Prácticos

- Documentación:
  - [MDN Web Docs - Event Reference](https://developer.mozilla.org/en-US/docs/Web/Events)
- Ejercicios:
  1. Crear una aplicación que cambie el color de fondo de una página cada vez que se haga clic en un botón.
  2. Implementar un juego de memoria donde los elementos cambian de posición al hacer clic.
  3. Crear un formulario con validación dinámica que muestre mensajes de error específicos en cada campo.

Con este contenido, los estudiantes podrán comprender a fondo el manejo de eventos en JavaScript y aplicarlos en el desarrollo de aplicaciones web dinámicas e interactivas.