Programador JavaScript Avanzado

Unidad 2: Eventos y Formularios

Contactos: tecnologia@elearning-total.com
Web: www.elearning-total.com

Indice

Unidad 2: Eventos y Formularios

Objetivos

Que el alumno logre:

- Conocer e implementar Eventos y Callbacks

Eventos y CallBacks

En JavaScript, los eventos y los callbacks son conceptos fundamentales que se utilizan para manejar la interacción con el usuario y ejecutar código en respuesta a acciones específicas.

Los eventos son acciones que ocurren en un elemento HTML, como hacer clic en un botón, mover el mouse sobre un elemento, cargar una página, etc. En JavaScript, podemos "*escuchar*" estos eventos y ejecutar código en respuesta a ellos.

Existen diferentes tipos de eventos, como eventos de clic (click), eventos de teclado (keydown, keyup), eventos de mouse (mouseover, mouseout), eventos de carga de página (load), entre otros.

Podemos asociar funciones a estos eventos para que se ejecuten cuando ocurra el evento correspondiente.

Partes del manejo de eventos:

- ***Event:*** Representa un evento en la aplicación.
- ***Event Listener:*** Es una función asociada a un evento. Al dispararse un evento, se disparan todos los Event Listeners asociados.
- ***Event Target:*** Es el lugar en donde ocurre el evento. Es la entidad encargada de disparar el evento y registrar los Event Listeners.

El DOM de JavaScript es un mapeo de todos los atributos HTML como objetos JavaScript. De esta forma, contamos con todos los atributos HTML para asignarlos de forma programada con JavaScript.

Entre estos atributos están los manejadores de eventos, que empiezan con el prefijo on. Asignar los on, mediante el DOM, sirve para separar mejor el código, pero sobre todo para asignar eventos de forma programada.

Atributos HTML On

Son atributos estándar HTML para el registro de eventos. Sirven para asociar la ejecución de una función a un evento que ocurra en la página (como el click de un botón o la escritura de un campo de texto).

OnClick

```
<a onclick="hacerClick()">Pulsar</a>
function hacerClick() {
     alert("Pulsaste el botón!")
}
```

OnInput

```
<input type="text" oninput="ingresoDato()" id="nombre">
function ingresoDato() {
          const input = document.getElementById('nombre')
          const valor = input.value
          console.log(valor)
        }
```

OnChange

```
 <select id="pais" onchange="elegirPais()">
      <option value="Argentina">Argentina</option>
      <option value="Uruguay">Uruguay</option>
      <option value="Brasil">Brasil</option>
      <option value="Chile">Chile</option>
    </select>

function elegirPais() {
      const select = document.getElementById('pais')
      const valor = select.value
      alert ('Elegiste el país: '+valor)
    }
```

Callbacks

Un callback en JavaScript es una función que se pasa como argumento a otra función y se ejecuta después de que la función original haya terminado su ejecución. En otras palabras, el callback es una forma de indicar a una función qué hacer después de que se hayan completado ciertas tareas o eventos.

Los callbacks son muy útiles en situaciones en las que se requiere trabajar con código asíncrono, donde una operación puede llevar tiempo y no queremos bloquear la ejecución del programa mientras esperamos su finalización. En lugar de esperar activamente a que se complete una tarea, podemos pasar una función de callback a la función asincrónica y permitir que esta última la invoque cuando esté lista.

Podemos implementar callbacks en distintas funcionalidades de nuestras aplicaciones como:

- ***Operaciones de lectura/escritura en archivos o bases de datos:*** Cuando se leen o escriben datos desde o hacia un archivo o una base de datos, estas operaciones suelen ser asíncronas. Los callbacks se utilizan para manejar los resultados o errores de estas operaciones una vez que se han completado.
- ***Solicitudes AJAX:*** Al realizar solicitudes a servidores para obtener o enviar datos, las respuestas pueden llevar tiempo. Los callbacks se utilizan para manejar la respuesta del servidor una vez que se reciba.
- ***Animaciones y transiciones:*** Cuando se realizan animaciones o transiciones en una página web, los callbacks se utilizan para ejecutar ciertas acciones una vez que la animación o la transición haya finalizado.
- ***Eventos del usuario:*** Cuando se producen eventos del usuario, como hacer clic en un botón o enviar un formulario, los callbacks se utilizan para ejecutar el código que responde a esos eventos.
- ***Temporizadores:*** Al usar funciones como setTimeout o setInterval, se puede proporcionar un callback que se ejecute después de que haya transcurrido un cierto período de tiempo o en intervalos regulares.

Los callbacks permiten estructurar y controlar el flujo de ejecución de código asíncrono de manera más clara y eficiente. Además, también facilitan la reutilización de código, ya que puedes pasar diferentes callbacks a una misma función según tus necesidades.

Es importante tener en cuenta que, con el advenimiento de JavaScript moderno, han surgido otros enfoques para manejar la asincronía, como Promises y async/await, que ofrecen una sintaxis más limpia y una gestión más robusta de los errores. Sin embargo, los callbacks siguen siendo una parte fundamental de JavaScript y es esencial comprender cómo funcionan.

Veamos un ejemplo:

```
function validarCondicion(valor, condicion, callback) {
  if (condicion(valor)) {
    callback(true);
  } else {
    callback(false);
  }
}

function mostrarValidacion(resultado) {
  if (resultado) {
    alert('La condición se cumple');
  } else {
    alert('La condición no se cumple');
  }
}

function esMayor (num) {
  return num > 18;
}

validarCondicion(15, esMayor, mostrarValidacion);
```

En este ejemplo, tenemos una función llamada **validarCondicion** que tiene como parámetros: un valor, una función de condición y un callback.

Dentro de la función, se evalúa si la condición se cumple para el valor proporcionado. Si la condición es verdadera, se llama al callback con true como argumento; de lo contrario, se llama al callback con false.

Además, definimos una función llamada **mostrarValidacion** que muestra un mensaje de alerta según el resultado de la validación.

También tenemos una función llamada **esMayor** que toma un número y devuelve true si es mayor a 18, y false en caso contrario.

Finalmente, llamamos a validarCondicion pasando el número 15, la función esMayor como condición, y mostrarValidacion como el callback.

Cuando se ejecuta el código, validarCondicion evalúa si el número 15 cumple la condición de ser mayor a 18. Como no es así, se llama a mostrarValidacion con false como argumento, y se muestra el mensaje "La condición no se cumple".

En este ejemplo podemos ver cómo se puede utilizar un callback para recibir y manejar el resultado de una validación basada en una condición específica.

Resumen

En esta Unidad…

Trabajamos con la Eventos y formularios

En la próxima Unidad…

Trabajaremos con AJAX