Tema 8
JavaScript

Objetivos

- Comprender los conceptos fundamentales de la programación en JavaScript, incluyendo la sintaxis básica y la estructura de un programa.
- Dominar el manejo de variables, tipos de datos y operadores en JavaScript para realizar cálculos y almacenar información de manera eficiente.
- Aprender las estructuras de control en JavaScript, incluyendo condicionales y bucles, para tomar decisiones y repetir acciones según las necesidades del programa.
- Explorar los eventos en JavaScript y cómo gestionarlos para interactuar con elementos del DOM, permitiendo la creación de aplicaciones interactivas.
- Comprender la importancia del Document Object Model (DOM) y cómo acceder a elementos en una página web utilizando JavaScript.
- Aprender a modificar el contenido y los atributos de elementos del DOM, así como crear y eliminar elementos dinámicamente para manipular la estructura de la página web.

Introducción

Tras haber explorado las bases de HTML y CSS, ahora nos adentraremos en el emocionante mundo de JavaScript. Este lenguaje de programación, ampliamente utilizado en el desarrollo web, nos permitirá dar vida a nuestras páginas web al agregar interactividad y dinamismo. A lo largo de esta sección, aprenderemos cómo JavaScript se integra con HTML y CSS para crear experiencias web más ricas y atractivas, permitiéndonos controlar el comportamiento de los elementos de la página y responder a las acciones del usuario.

Índice

8.1 INTRODUCCIÓN A LA PROGRAMACIÓN EN JAVASCRIPT

JavaScript es un lenguaje de programación que desempeña un papel esencial en la creación de experiencias web interactivas y dinámicas. Cuando exploramos la programación en el contexto de la informática y la web, JavaScript destaca como una de las herramientas más influyentes y omnipresentes.

¿Qué es JavaScript?

JavaScript, a menudo abreviado como JS, es un lenguaje de programación de alto nivel, interpretado y **orientado a objetos**. A diferencia de HTML (HyperText Markup Language) y CSS (Cascading Style Sheets), que se utilizan principalmente para estructurar y dar estilo a las páginas web, **JavaScript se utiliza para agregar interactividad y comportamientos dinámicos a las páginas web.**
Aunque JavaScript comparte su nombre con Java, otro lenguaje de programación, no tienen mucho en común en términos de sintaxis o funcionalidad. JavaScript fue diseñado específicamente para la web y se ejecuta en el navegador web del usuario final.

Importancia de JavaScript en la Tripleta HTML+CSS+JS

La tripleta HTML+CSS+JS es el núcleo de la programación web moderna. Aquí está la razón por la cual cada uno de estos componentes es crucial:

1. **HTML (HyperText Markup Language)**: Proporciona la estructura y el contenido de una página web. Define los elementos y su disposición en la página.

2. **CSS (Cascading Style Sheets)**: Controla la presentación y el estilo de la página web. Determina cómo se verán los elementos HTML.

3. **JavaScript**: Agrega interactividad y comportamientos a la página. Permite que los elementos HTML y CSS respondan a las acciones del usuario y realicen operaciones complejas en el lado del cliente.

La combinación de estos tres elementos permite crear sitios web modernos y atractivos que van más allá de la simple presentación de información estática. JavaScript desempeña un papel crucial al permitirnos crear aplicaciones web interactivas, juegos, formularios dinámicos y muchas otras características que mejoran la experiencia del usuario.

Importancia de JavaScript en el Desarrollo Web

La importancia de JavaScript en el desarrollo web es innegable y se deriva de varias características clave:

1. **Interactividad del Usuario**: JavaScript permite a los desarrolladores crear aplicaciones web interactivas que responden a las acciones del usuario. Esto incluye formularios que validan datos en tiempo real, menús desplegables, carruseles de imágenes y mucho más.

2. **Manipulación del DOM**: El Document Object Model (DOM) representa la estructura de una página web y JavaScript permite a los desarrolladores acceder y manipular dinámicamente el DOM. Esto significa que se pueden agregar, eliminar o modificar elementos HTML y CSS en respuesta a eventos o acciones del usuario.

3. **Comunicación con el Servidor**: JavaScript facilita las solicitudes asíncronas al servidor a través de la tecnología AJAX (Asynchronous JavaScript and XML), lo que permite cargar datos en segundo plano sin recargar la página completa. Esto se utiliza ampliamente en aplicaciones web modernas para mejorar la velocidad y la eficiencia.

4. **Amplio Ecosistema**: Existe una vasta colección de bibliotecas y marcos de trabajo (como jQuery, React, Angular y Vue.js) construidos sobre JavaScript que facilitan el desarrollo web. Estas herramientas aceleran el proceso de desarrollo y ayudan a crear aplicaciones web avanzadas de manera eficiente.

El Poder de JavaScript en Acción

Cuando dominamos JavaScript, podemos crear una amplia variedad de aplicaciones y efectos web:

1. **Validación de Formularios en Tiempo Real**: Verificar datos ingresados en formularios antes de su envío.

2. **Animaciones y Transiciones Suaves**: Crear animaciones y efectos visuales atractivos.

3. **Carruseles de Imágenes Interactivos**: Desarrollar componentes deslizantes de imágenes dinámicos.

4. **Juegos en Línea**: Construir juegos web simples y entretenidos.

5. **Actualizaciones de Contenido en Tiempo Real**: Cargar datos y actualizar la página sin necesidad de recargarla.

6. **Aplicaciones Web Complejas**: Crear aplicaciones web complejas con funcionalidad de usuario avanzada.

Dado el corto espacio de tiempo que se dispone en este curso, durante la primera parte del tema estudiaremos conceptos básicos de programación en concreto en javaScript y seguidamente exploraremos las capacidades de JavaScript en lo que a **manipulación del DOM** se refiere y aprenderemos a utilizar este poderoso lenguaje para crear experiencias web ricas e interactivas.

8.2 VARIABLES, TIPOS DE DATOS Y OPERADORES

En este segundo apartado, nos sumergiremos en los fundamentos de JavaScript, explorando la declaración de variables, los tipos de datos disponibles y cómo utilizar operadores aritméticos y lógicos. Estos conceptos son esenciales para comprender cómo funciona JavaScript y cómo se pueden realizar operaciones y manipulaciones de datos en el lenguaje.

Declaración de Variables en JavaScript

Las variables son elementos fundamentales en cualquier lenguaje de programación, y JavaScript no es la excepción. Las variables se utilizan para almacenar y gestionar datos en un programa. En JavaScript, podemos declarar variables utilizando las palabras clave **let** y **const**.

```
// Usando let
let saldo = 1000;
let mensaje = "¡Hola, mundo!";

// Usando const (para valores constantes)
const PI = 3.14159265359;
const URL_API = "https://api.example.com";
```

Tipos de Datos en JavaScript

JavaScript admite diversos tipos de datos que se utilizan para representar diferentes tipos de información. Los tipos de datos más comunes en JavaScript son:

1. **Números (Numbers)**: Representan valores numéricos, ya sean enteros o decimales. Ejemplo: **let edad = 25;**

2. **Cadenas de Texto (Strings)**: Almacenan texto. Ejemplo: **let nombre = "María";**

3. **Booleanos (Booleans)**: Representan valores de verdad (**true**) o falsedad (**false**). Ejemplo: **let activado = true;**

4. **Vectores (Arrays)**: Almacenan colecciones ordenadas de datos. Ejemplo: **let colores = ["rojo", "verde", "azul"];**

5. **Objetos (Objects)**: Almacenan colecciones de pares clave-valor. Ejemplo:

```
let persona = {
    nombre: "Juan",
    edad: 30,
    casado: false
};
```

Operadores Aritméticos y Lógicos

Los operadores son herramientas fundamentales en programación para realizar cálculos y comparaciones. En JavaScript, tenemos una variedad de operadores a nuestra disposición:

- **Operadores Aritméticos**: Se utilizan para realizar cálculos matemáticos, como suma, resta, multiplicación, división, módulo, incremento y decremento. Ejemplo:

```
let suma = 5 + 3;
let resta = 10 - 4;
let multiplicacion = 6 * 7;
let division = 20 / 4;
let modulo = 20 % 4; //Da como resultado 0 que es el resto de dividir 20 entre 4
let division = 20 / 4;
```

- **Operadores de Comparación**: Permiten comparar valores y devuelven un valor booleano (**true** o **false**). Ejemplo:

```
let esMayor = 10 > 5; // true
let esIgual = 5 === "5"; // false (comparación estricta)
```

- **Operadores Lógicos**: Se utilizan para combinar expresiones booleanas y realizar operaciones lógicas como AND (**&&**), OR (**||**) y NOT(!). Ejemplo:

```
let condicion1 = true;
let condicion2 = false;

let resultadoAND = condicion1 && condicion2; // false
let resultadoOR = condicion1 || condicion2; // true
```

En esta sesión, exploraremos más ejemplos prácticos de cómo utilizar estas variables, tipos de datos y operadores para realizar cálculos y tomar decisiones en programas JavaScript. Estos conceptos son la base para construir algoritmos y aplicaciones más complejas en sesiones posteriores del curso.

&& (AND)		
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

|| (OR)		
True	True	**True**
True	False	**True**
False	True	**True**
False	False	**False**

!(NOT)		
True	True	**True**
True	False	**True**

```
// Ejemplo 1: Declaración de variables
let nombre = "Juan";
let edad = 25;
// Ejemplo 2: Tipos de datos
let mensaje = "¡Hola, mundo!";
let saldo = 1000;
let activado = true;
// Ejemplo 3: Operadores aritméticos
let suma = 5 + 3; // Resultado: 8
```

```
let resta = 10 - 4; // Resultado: 6
let multiplicacion = 6 * 7; // Resultado: 42
let division = 20 / 4; // Resultado: 5
// Ejemplo 4: Operadores de comparación
let esMayor = edad > 18; // Resultado: true
let esIgual = edad === 25; // Resultado: true
let esDiferente = saldo !== 0; // Resultado: true
// Ejemplo 5: Operadores lógicos
let condicion1 = true;
let condicion2 = false;
let resultadoAND = condicion1 && condicion2; // Resultado: false
let resultadoOR = condicion1 || condicion2; // Resultado: true
// Ejemplo 6: Concatenación de cadenas de texto
let nombreCompleto = nombre + " Pérez"; // Resultado: "Juan Pérez"
// Ejemplo 7: Modificación de variables
edad = 26; // Nueva edad: 26
// Ejemplo 8: Uso de comillas simples y dobles en cadenas
let mensaje2 = 'Ella dijo: "Hola"'; // Resultado: 'Ella dijo: "Hola"'
// Ejemplo 9: Asignación de variables a otras variables
let x = 5;
let y = x; // y tendrá el valor de 5
// Ejemplo 10: Operador de incremento
let contador = 0;
contador++; // contador ahora es 1
// Ejemplo 11: Operador de decremento
let contador2 = 10;
contador2--; // contador2 ahora es 9
// Ejemplo 12: Concatenación de cadenas y números
let numero = 42;
let texto = "El número es: " + numero; // Resultado: "El número es: 42"
// Ejemplo 13: Conversión de tipos de datos
let numeroString = "10";
let numeroEntero = parseInt(numeroString); // Resultado: 10
// Ejemplo 14: Operaciones mixtas
let resultadoMixto = 5 + "5"; // Resultado: "55"
// Ejemplo 15: Uso de operadores de asignación
let cantidad = 5;
cantidad += 3; // cantidad ahora es 8
// Ejemplo 16: Comparación de valores
let comparacion = (edad > 30); // Resultado: false
// Ejemplo 17: Operador ternario
let esMayorEdad = (edad >= 18) ? "Mayor de edad" : "Menor de edad";
// Ejemplo 18: Concatenación de cadenas con template literals (plantillas literales)
let producto = "Manzanas";
```

```
let cantidadProducto = 10;
let mensajeProducto = `Tienes ${cantidadProducto} ${producto}.`; // Resultado: "Tienes 10
Manzanas."
// Ejemplo 19: Uso de NaN (Not-a-Number)
let resultadoNaN = 10 / "texto"; // Resultado: NaN
// Ejemplo 20: Uso de typeof para verificar tipos de datos
let tipoEdad = typeof edad; // Resultado: "number"
```

8.3 ESTRUCTURAS DE CONTROL.

En esta tercera sesión, exploraremos las estructuras de control en JavaScript, las declaraciones condicionales y las sentencias repetitivas o bucles, que nos permiten tomar decisiones en función de condiciones específicas o realizar repeticiones según el caso. A lo largo de esta sesión, aprenderemos a utilizar las declaraciones **if**, **if anidado** y **else**, **while**, **do..while** y **for**.

Las declaraciones condicionales son fundamentales en la programación, ya que nos permiten controlar el flujo de ejecución de un programa en función de una condición. En JavaScript, utilizamos las siguientes estructuras condicionales:

if: La declaración **if** permite ejecutar un bloque de código si una condición es verdadera.

```
let edad = 18;

if (edad >= 18) {
    console.log("Eres mayor de edad.");
}
```

If anidado: La declaración **else if** se utiliza para evaluar una condición adicional si la primera condición **if** es falsa.

```
let hora = 15;

if (hora < 12) {
    console.log("Buenos días.");
} else if (hora < 18) {
    console.log("Buenas tardes.");
} else {
    console.log("Buenas noches.");
}
```

else: La declaración **else** se ejecuta si es false la condición del if.

```
let saldo = 0;

if (saldo > 0) {
    console.log("Tienes saldo positivo.");
} else {
    console.log("No tienes saldo.");
}
```

Ejemplos Prácticos

Ahora, realizaremos algunos ejercicios prácticos para aplicar lo que hemos aprendido sobre declaraciones condicionales:

Ejercicio 1: Verificación de Contraseña

```
let contraseña = "secreta";
let contraseñaUsuario = "secreta123";

if (contraseñaUsuario === contraseña) {
    console.log("Contraseña correcta.");
} else {
    console.log("Contraseña incorrecta.");
}
```

Ejercicio 2: Clasificación de Números

```
let numero = 15;

if (numero > 0) {
    console.log("El número es positivo.");
} else if (numero < 0) {
    console.log("El número es negativo.");
} else {
    console.log("El número es igual a cero.");
}
```

Ejercicio 3: Determinación de Día de la Semana

```
let dia = "martes";

if (dia === "lunes") {
    console.log("Hoy es el primer día laborable de la semana.");
} else if (dia === "viernes") {
    console.log("¡Viernes! Fin de la semana laborable.");
} else {
    console.log("Estamos en mitad de la semana laborable.");
}
```

Bucles en JavaScript

Los bucles son fundamentales en la programación, ya que nos permiten realizar tareas repetitivas de manera eficiente. En JavaScript, utilizamos las siguientes declaraciones de bucles:

for: La declaración **for** se utiliza para ejecutar un bloque de código un número específico de veces. Generalmente, se utiliza cuando conocemos la cantidad exacta de iteraciones que deseamos.

```
for (let i = 0; i < 5; i++) {
    console.log("Iteración " + i);
}
```

while: La declaración **while** ejecuta un bloque de código mientras una condición sea verdadera. Se usa cuando no sabemos cuántas veces se ejecutará el bucle de antemano.

```
let contador = 0;

while (contador < 3) {
    console.log("Iteración " + contador);
    contador++;
}
```

do...while: La declaración **do...while** es similar a **while**, pero garantiza que el bloque de código se ejecutará al menos una vez antes de verificar la condición.

```
let intentos = 0;

do {
    console.log("Intento #" + intentos);
    intentos++;
} while (intentos < 3);
```

Ejemplos Prácticos con Bucles

Aquí tienes algunos ejercicios prácticos para aplicar los conceptos de bucles en JavaScript:

Ejercicio 1: Imprimir Números Pares

```
for (let i = 0; i <= 10; i += 2) {
    console.log(i);
}
```

Ejercicio 2: Sumar Números del 1 al 10

```
let suma = 0;

for (let i = 1; i <= 10; i++) {
    suma += i;
}
console.log("La suma de los números del 1 al 10 es: " + suma);
```

Ejercicio 3: Adivinar un Número

```
const numeroAleatorio = Math.floor(Math.random() * 10) + 1;
let intentos = 0;
let adivinanza;

do {
    adivinanza = prompt("Adivina el número (entre 1 y 10):");
```

```
    intentos++;
    if (parseInt(adivinanza) === numeroAleatorio) {
        console.log("¡Correcto! El número era " + numeroAleatorio);
        break;
    } else {
        console.log("Intento #" + intentos + ": Intenta de nuevo.");
    }
} while (intentos < 3);

if (intentos === 3) {
    console.log("Agotaste tus intentos. El número era " + numeroAleatorio);
}
```

8.4 FUNCIONES Y ÁMBITO DE VARIABLES EN JAVASCRIPT

En esta sección, exploraremos el concepto de funciones en JavaScript y cómo afectan al ámbito (scope) de las variables. Las funciones son bloques de código reutilizables que realizan tareas específicas y son fundamentales en la programación JavaScript.

Funciones en JavaScript

Una función es un bloque de código que se puede llamar y ejecutar en cualquier parte de un programa. Las funciones ayudan a organizar y reutilizar el código de manera eficiente. Para definir una función en JavaScript, puedes utilizar la siguiente sintaxis:

```
function nombreDeLaFuncion(parametro1, parametro2) {
    // Código de la función
    return resultado;
}
```

- **nombreDeLaFuncion**: Es el nombre que eliges para la función.
- **parametro1**, **parametro2**: Son los valores que la función puede recibir como entrada (argumentos).
- **return resultado**: La función puede devolver un resultado opcional.

```
function saludar(nombre) {
    return "Hola, " + nombre + "!";
}
let mensaje = saludar("Juan");
console.log(mensaje); // Resultado: "Hola, Juan!"
```

Ámbito de Variables (Scope)

El ámbito se refiere a la visibilidad y accesibilidad de las variables en un programa. En JavaScript, existen dos tipos principales de ámbito de variables:

Ámbito Global: Las variables globales se declaran fuera de cualquier función y son accesibles desde cualquier parte del programa.

```
let variableGlobal = "Soy global";

function funcion1() {
    console.log(variableGlobal); // Puedo acceder a la variable global
}
```

Ámbito Local o de Función: Las variables locales se declaran dentro de una función y solo son accesibles desde esa función.

```
function funcion2() {
    let variableLocal = "Soy local";
    console.log(variableLocal); // Puedo acceder a la variable local
}
```

Funciones Anónimas y Expresiones de Funciones

Además de declarar funciones con un nombre, también puedes crear funciones anónimas y expresiones de funciones. Estas se utilizan comúnmente como argumentos de otras funciones o para definir funciones dentro de objetos.

Función Anónima:

```
let sumar = function (a, b) {
    return a + b;
};

let resultado = sumar(5, 3); // Llamando a la función anónima
```

Expresión de Función:

```
let multiplicar = function multiplica(a, b) {
    return a * b;
};

let producto = multiplicar(4, 6); // Llamando a la expresión de función
```

En resumen, las funciones son una parte esencial de JavaScript que te permite modular y reutilizar tu código de manera efectiva. Comprendiendo el ámbito de las variables y cómo funcionan las funciones, estarás mejor preparado para escribir programas JavaScript más estructurados y mantenibles.

8.5 . ARRAYS Y OBJETOS EN JAVASCRIPT

En esta sección, exploraremos los conceptos de arrays y objetos en JavaScript. Estas son estructuras de datos fundamentales que se utilizan para almacenar y organizar información de manera eficiente.

Arrays en JavaScript

Un array es una colección ordenada de elementos. Cada elemento en un array tiene un índice numérico que comienza en 0. Los arrays pueden contener elementos de cualquier tipo, incluyendo números, cadenas, objetos u otros arrays. Para crear un array en JavaScript, puedes utilizar la siguiente sintaxis:

```
let miArray = [elemento1, elemento2, elemento3];
```

Ejemplo de un Array:

```
let colores = ["rojo", "verde", "azul"];
```

Para acceder a elementos individuales de un array, puedes utilizar su índice:

```
console.log(colores[0]); // Resultado: "rojo"
```

Objetos en JavaScript

Un objeto es una estructura de datos que almacena pares clave-valor. Cada valor está asociado a una clave única que actúa como identificador. Los objetos son útiles para representar entidades del mundo real y sus propiedades. Para crear un objeto en JavaScript, puedes utilizar la siguiente sintaxis:

```
let miObjeto = {
    clave1: valor1,
    clave2: valor2,
    clave3: valor3
};
```

Ejemplo de un objeto.

```
let persona = {
    nombre: "Juan",
    edad: 30,
    casado: false
};
```

Para acceder a las propiedades de un objeto, puedes utilizar la notación de punto o la notación de corchetes:

```
console.log(persona.nombre); // Resultado: "Juan"
console.log(persona["edad"]); // Resultado: 30
```

Arrays de Objetos

Los arrays y objetos se pueden combinar para crear estructuras de datos más complejas. Por ejemplo, puedes tener un array de objetos que representen elementos similares:

```
let estudiantes = [
```

```
    { nombre: "María", edad: 25 },
    { nombre: "Carlos", edad: 22 },
    { nombre: "Ana", edad: 30 }
];
```

Iteración a través de Arrays y Objetos

Puedes utilizar bucles como **for** o **forEach** para iterar a través de los elementos de un array o las propiedades de un objeto:

Iteración a través de un Array:

```
for (let i = 0; i < colores.length; i++) {
    console.log(colores[i]);
}
```

Iteración a través de un Objeto:

```
for (let clave in persona) {
    console.log(clave + ": " + persona[clave]);
}
```

Los arrays y objetos son estructuras de datos cruciales en JavaScript que te permiten almacenar y organizar información de manera efectiva. Los arrays son ideales para colecciones de elementos similares, mientras que los objetos son útiles para representar entidades y sus propiedades. Comprender cómo trabajar con estas estructuras de datos es esencial para el desarrollo web en JavaScript.

8.6 . EVENTOS Y GESTIÓN DE ERRORES EN JAVASCRIPT.

En esta sección, abordaremos los conceptos de eventos y la gestión de errores en javascript. los eventos son interacciones del usuario o eventos del navegador que desencadenan acciones en tu aplicación web. La gestión de errores es fundamental para identificar, manejar y solucionar problemas en tu código.

Eventos en JavaScript

Los eventos son acciones que ocurren en una página web, como hacer clic en un botón, mover el mouse, escribir en un campo de texto, etc. JavaScript permite la captura y manipulación de estos eventos para crear aplicaciones web interactivas.

Para agregar un evento a un elemento HTML, primero debes seleccionar ese elemento y luego asignarle una función que se ejecutará cuando ocurra el evento.

Ejemplo de Evento Click:

```
// Selección del elemento con id "miBoton"
```

```
let boton = document.getElementById("miBoton");

// Agregar un evento de clic al botón
boton.addEventListener("click", function() {
    alert("¡Hiciste clic en el botón!");
});
```

He aquí una tabla con los principales eventos en javascript.

Evento	Descripción
click	Se dispara cuando se hace clic en un elemento.
mouseover	Ocurre cuando el cursor del mouse entra en un elemento.
mouseout	Ocurre cuando el cursor del mouse sale de un elemento.
keydown	Se dispara cuando una tecla del teclado se presiona.
keyup	Se dispara cuando se suelta una tecla del teclado.
submit	Se dispara cuando se envía un formulario.
change	Ocurre cuando el valor de un elemento de entrada cambia (como un campo de texto o una casilla de verificación).
load	Se dispara cuando se completa la carga de un recurso, como una imagen o un archivo de script.
DOMContentLoaded	Ocurre cuando el documento HTML se ha cargado completamente y está listo para ser manipulado mediante JavaScript.
resize	Se dispara cuando se cambia el tamaño de la ventana del navegador.
scroll	Ocurre cuando el usuario desplaza la página web.
focus	Se dispara cuando un elemento obtiene el foco (como un campo de entrada).
blur	Se dispara cuando un elemento pierde el foco.

Gestión de Errores en JavaScript

La gestión de errores es crucial para identificar y manejar problemas que puedan surgir en tu código. JavaScript proporciona bloques **try...catch** para capturar y manejar excepciones.

Ejemplo de Captura de Errores:

```
try {
    // Código que puede generar un error
    let resultado = 10 / 0;
} catch (error) {
    // Manejo del error
    console.log("Se produjo un error: " + error.message);
}
```

La captura de errores permite que tu aplicación continúe ejecutándose incluso si se produce un error, en lugar de detenerse por completo.

Manejo de Errores Personalizados:

Puedes crear tus propias excepciones personalizadas utilizando el objeto **Error** y lanzarlas en tu código cuando sea necesario:

```
function dividir(a, b) {
    if (b === 0) {
        throw new Error("No se puede dividir por cero.");
    }
    return a / b;
}

try {
    let resultado = dividir(10, 0);
} catch (error) {
    console.log("Error: " + error.message);
}
```

Eventos y Gestión de Errores en Práctica:

Puedes combinar eventos y gestión de errores para crear aplicaciones web más robustas y con mejor capacidad de respuesta. Por ejemplo, puedes manejar errores en formularios web para proporcionar retroalimentación al usuario sobre entradas inválidas.

```
let formulario = document.getElementById("miFormulario");

formulario.addEventListener("submit", function(event) {
    try {
        // Validación del formulario
        if (formulario.nombre.value === "") {
            throw new Error("El campo de nombre no puede estar vacío.");
        }
        // Resto del proceso de envío del formulario
    } catch (error) {
        event.preventDefault(); // Evitar el envío del formulario
        alert("Error: " + error.message);
    }
});
```

En resumen, los eventos y la gestión de errores son aspectos esenciales en el desarrollo web con JavaScript. Los eventos permiten que tu aplicación responda a las acciones del usuario, mientras que la gestión de errores te ayuda a identificar y manejar problemas en tu código de manera efectiva, mejorando la confiabilidad y la usabilidad de tus aplicaciones web.

8.7. INTRODUCCIÓN AL DOCUMENT OBJECT MODEL (DOM) EN JAVASCRIPT

El Document Object Model, o DOM, es una representación programática de la estructura de un documento HTML o XML. En otras palabras, el DOM permite a los desarrolladores acceder, manipular y modificar los elementos y contenido de una página web utilizando JavaScript.

Estructura del DOM

El DOM organiza la estructura del documento en una jerarquía de nodos. Los nodos representan partes del documento, como elementos HTML, atributos y texto. Los nodos se organizan en un árbol, con un nodo raíz llamado "documento". Aquí hay una breve descripción de algunos tipos de nodos comunes:

1. **Elementos**: Representan etiquetas HTML, como ** **, **<p>**, **<h1>**, etc.
2. **Atributos**: Son los valores asociados a los atributos de los elementos, como **id**, **class**, **src**, etc.
3. **Texto**: Representa el contenido de texto dentro de un elemento HTML.
4. **Nodos Padre e Hijos**: Los elementos pueden tener nodos hijos (elementos, atributos o texto) y un nodo padre (el elemento que contiene a los hijos).

Manipulación del DOM en JavaScript

JavaScript te permite acceder y manipular el DOM de una página web. Puedes seleccionar elementos, cambiar su contenido, modificar atributos y responder a eventos del usuario. Aquí tienes algunos ejemplos:

Seleccionar Elementos:

```
// Seleccionar un elemento por su id→ getElementById
let miElemento = document.getElementById("miId");

// Seleccionar varios elementos por su clase→ getElementsByClassName
let elementos = document.getElementsByClassName("miClase");

// Seleccionar elementos por su etiqueta→ getElementsByTagName
let parrafos = document.getElementsByTagName("p");
```

Cambiar el Contenido de un Elemento: (innerHTML)

```
let miParrafo = document.getElementById("parrafo");
miParrafo.innerHTML = "Nuevo contenido del párrafo";
```

Modificar Atributos:

```
let miImagen = document.getElementById("imagen");
miImagen.src = "nueva-imagen.jpg";
```

Responder a Eventos: `addEventListener`

```
let miBoton = document.getElementById("boton");
miBoton.addEventListener("click", function() {
    alert("Hiciste clic en el botón");
});
```

Crear Nuevos Elementos: `appendChild`

```
let nuevoParrafo = document.createElement("p");
nuevoParrafo.innerHTML = "Este es un nuevo párrafo";
document.body.appendChild(nuevoParrafo);
```

Eliminar Elementos: `removeChild`

```
let elementoAEliminar = document.getElementById("elemento");
elementoAEliminar.parentNode.removeChild(elementoAEliminar);
```

Beneficios del DOM

- **Interactividad**: Permite crear aplicaciones web interactivas que respondan a las acciones del usuario.
- **Accesibilidad**: Facilita la manipulación y actualización de contenido web en tiempo real.
- **Dinamismo**: Permite modificar la apariencia y el contenido de una página sin necesidad de recargarla.

DOM es una parte esencial del desarrollo web en JavaScript. Proporciona una interfaz para acceder y modificar elementos HTML, lo que permite crear experiencias interactivas y dinámicas en aplicaciones web. Comprender cómo funciona el DOM es fundamental para cualquier desarrollador web front-end.

Ejemplo 1: Cambiar el Texto de un Párrafo

Dado el siguiente código HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejercicio 1</title>
</head>
<body>
    <p id="miParrafo">Este es un párrafo de ejemplo.</p>
    <button id="miBoton">Cambiar Texto</button>

    <script src="script1.js"></script>
```

```
</body>
</html>
```

scrip1.js

```
// Seleccionar el párrafo y el botón
let miParrafo = document.getElementById("miParrafo");
let miBoton = document.getElementById("miBoton");

// Agregar un evento de clic al botón
miBoton.addEventListener("click", function() {
    // Cambiar el texto del párrafo
    miParrafo.innerHTML = "¡Texto cambiado!";
});
```

Cuando hagas clic en el botón, el texto del párrafo se cambiará a "¡Texto cambiado!".

Ejercicio 2: Crear Elementos Dinámicamente

En este ejercicio, crearás un nuevo elemento de lista (li) y lo agregarás a una lista (ul) cuando se haga clic en un botón.

HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejercicio 2</title>
</head>
<body>
    <ul id="miLista">
        <li>Elemento 1</li>
        <li>Elemento 2</li>
    </ul>
    <button id="miBoton">Agregar Elemento</button>

    <script src="script2.js"></script>
</body>
</html>
```

script2.js

```
// Seleccionar la lista y el botón
let miLista = document.getElementById("miLista");
let miBoton = document.getElementById("miBoton");

// Agregar un evento de clic al botón
miBoton.addEventListener("click", function() {
```

```
    // Crear un nuevo elemento de lista
    let nuevoElemento = document.createElement("li");
    nuevoElemento.innerHTML = "Nuevo Elemento";

    // Agregar el nuevo elemento a la lista
    miLista.appendChild(nuevoElemento);
});
```

Ejercicio 3: Cambiar el Color de Fondo

En este ejercicio, cambiarás el color de fondo de la página web cuando se haga clic en un botón.

HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejercicio 3</title>
</head>
<body>
    <button id="miBoton">Cambiar Color</button>

    <script src="script3.js"></script>
</body>
</html>
```

script3.js

```
// Seleccionar el botón
let miBoton = document.getElementById("miBoton");

// Agregar un evento de clic al botón
miBoton.addEventListener("click", function() {
    // Cambiar el color de fondo de la página
    document.body.style.backgroundColor = getRandomColor();
});

// Función para obtener un color aleatorio
function getRandomColor() {
    let letters = "0123456789ABCDEF";
    let color = "#";
    for (let i = 0; i < 6; i++) {
        color += letters[Math.floor(Math.random() * 16)];
    }
    return color;
}
```

8.8. MANIPULACIÓN DEL DOM.

Modificación de Contenido y Atributos de Elementos

La modificación del contenido y los atributos de elementos es fundamental para la actualización dinámica de una página web.

Ejemplo 1: Cambio del Atributo SRC de una Imagen

```
// Seleccionar una imagen por su id
let miImagen = document.getElementById("miImagen");

// Cambiar el atributo src de la imagen
miImagen.src = "nueva-imagen.jpg";
```

Ejemplo 2: Cambio de Texto de un Encabezado.

Supongamos que tienes el siguiente encabezado h1 en tu página HTML:

```
<h1 id="miTitulo">Título Original</h1>
```

Puedes utilizar JavaScript para cambiar el texto del encabezado cuando se carga la página:

```
// Seleccionar el encabezado h1 por su id
let miTitulo = document.getElementById("miTitulo");

// Cambiar el texto del encabezado
miTitulo.textContent = "Nuevo Título";
```

Ejemplo 3: Cambio de Estilo de un Párrafo

Supongamos que tienes el siguiente párrafo en tu página HTML:

```
<p id="miParrafo">Este es un párrafo de ejemplo.</p>
```

Puedes utilizar JavaScript para cambiar el contenido del párrafo de la siguiente manera:

```
// Seleccionar un párrafo por su id
let miParrafo = document.getElementById("miParrafo");

// Modificar el texto del párrafo
miParrafo.textContent = "Texto modificado del párrafo";
```

Ahora, el contenido del párrafo se ha cambiado a "Texto modificado del párrafo".

Ejemplo4: Cambio de Enlace de una Imagen

Supongamos que tienes una imagen en tu página HTML con un enlace original:

```
<img id="miImagen" src="imagen1.jpg" alt="Imagen Original">
```

Puedes utilizar JavaScript para cambiar el enlace (src) de la imagen cuando se hace clic en un botón:

```
// Seleccionar la imagen por su id
let miImagen = document.getElementById("miImagen");

// Seleccionar el botón por su id
let miBoton = document.getElementById("miBoton");

// Agregar un evento de clic al botón
miBoton.addEventListener("click", function() {
    // Cambiar el enlace (src) de la imagen
    miImagen.src = "imagen2.jpg";
    // Cambiar el atributo "alt" de la imagen
    miImagen.alt = "Nueva Imagen";
});
```

En este ejemplo, cuando hagas clic en el botón, el enlace de la imagen se cambiará a "imagen2.jpg" y el atributo "alt" se cambiará a "Nueva Imagen". Esto puede ser útil para cambiar dinámicamente el contenido de una imagen en respuesta a eventos del usuario, como un botón de "Cambiar Imagen".

Creación y Eliminación de Elementos en el DOM

JavaScript permite crear nuevos elementos HTML y agregarlos al DOM, así como eliminar elementos existentes.

Me remito a los ejemplos del punto anterior.

Gestión de Eventos en Elementos en el DOM

Ejemplo 1: Evento Click en un Botón

Supongamos que tienes un botón en tu página HTML y deseas mostrar un mensaje cuando el usuario haga clic en él.

HTML:

```
<button id="miBoton">Haz clic</button>
```

JavaScript

```
// Seleccionar el botón por su id
let miBoton = document.getElementById("miBoton");

// Agregar un evento de clic al botón
miBoton.addEventListener("click", function() {
    alert("Hiciste clic en el botón");
```

```
});
```

Cuando el usuario haga clic en el botón, aparecerá un cuadro de alerta con el mensaje "Hiciste clic en el botón".

Ejemplo 2: Evento Mouseover en una Imagen

Supongamos que tienes una imagen y deseas cambiar su tamaño cuando el usuario coloca el mouse sobre ella.

HTML:

```
<img id="miImagen" src="imagen.jpg" alt="Imagen">
```

JavaScript:

```
// Seleccionar la imagen por su id
let miImagen = document.getElementById("miImagen");

// Agregar un evento de mouseover a la imagen
miImagen.addEventListener("mouseover", function() {
    // Cambiar el ancho de la imagen al doble
    miImagen.style.width = "400px";
});
```

Cuando el usuario coloque el mouse sobre la imagen, esta se ampliará al doble de su tamaño original.

Ejemplo 3: Evento Keydown en un Campo de Texto

Supongamos que tienes un campo de texto y deseas mostrar un mensaje cuando el usuario presione una tecla en el campo.

HTML:

```
<input id="miInput" type="text" placeholder="Escribe algo">
```

JavaScript:

```
// Seleccionar el campo de texto por su id
let miInput = document.getElementById("miInput");

// Agregar un evento de keydown al campo de texto
miInput.addEventListener("keydown", function(event) {
    alert("Presionaste la tecla: " + event.key);
});
```

Cuando el usuario escriba algo en el campo de texto y presione una tecla, aparecerá un cuadro de alerta que mostrará la tecla presionada.