JavaScript en el desarrollo web

Miguel Angel Luna

Ya Basta! en la Secretaría de Asuntos Académicos

10 de agosto de 2024

Índice general

Capítulo 1

Variables, estructuras de control y funciones

1.1. Introducción a JavaScript

1.1.1. Breve reseña histórica de JavaScript

JavaScript, conocido también como JS, es uno de los lenguajes de programación más influyentes y ampliamente utilizados en el desarrollo web. Su historia comienza en 1995 cuando Brendan Eich, un ingeniero de Netscape Communications Corporation, creó el lenguaje en tan solo diez días. Originalmente llamado Mocha, luego LiveScript, y finalmente JavaScript, este lenguaje fue diseñado para agregar interactividad y dinamismo a las páginas web.

Inicialmente, JavaScript fue pensado como un lenguaje de scripting ligero, complementario a Java, que se ejecutaba en el lado del cliente (navegador web). Su sintaxis fue influenciada por lenguajes como C, Java, y Self, buscando ser accesible y fácil de aprender para los desarrolladores.

En sus primeros años, JavaScript fue subestimado y muchas veces criticado por sus limitaciones y problemas de compatibilidad entre diferentes navegadores. Sin embargo, su capacidad para manipular el Documento de Modelo de Objetos (DOM) y su versatilidad para crear efectos interactivos lo hicieron indispensable para el desarrollo web.

A medida que la web evolucionaba, también lo hizo JavaScript. La estandarización del lenguaje por la Asociación Europea de Fabricantes de Computadoras (ECMA) en 1997, bajo el nombre de ECMAScript, fue un paso crucial. La especificación ECMAScript proporcionó una base estándar para que los desarrolladores y los navegadores pudieran implementar y mejorar JavaScript de manera coherente.

El lanzamiento de ECMAScript 5 (ES5) en 2009 fue un hito importante, introduciendo características como el modo estricto, los métodos 'Array.prototype.forEach', 'Object.defineProperty', y más. Este estándar hizo que el lenguaje fuera más robusto y adecuado para aplicaciones más grandes y complejas.

El verdadero renacimiento de JavaScript llegó con ECMAScript 6 (ES6), también conocido como ECMAScript 2015. ES6 trajo consigo una gran cantidad de nuevas características y mejoras, como las clases, módulos, promesas, la sintaxis de flecha ('=>'), y 'let' y 'const' para el ámbito de bloque. Estas adiciones transformaron a JavaScript en un lenguaje moderno.

Con la llegada de Node.js en 2009, JavaScript trascendió el navegador y se convirtió en un lenguaje de propósito general. Node.js permitió a los desarrolladores utilizar JavaScript para escribir código del lado del servidor, manejando aplicaciones web completas con un único lenguaje de programación.

Hoy en día, las bibliotecas y frameworks como React, Angular, Vue.js, y muchos otros han ampliado aún más las posibilidades de JavaScript, facilitando el desarrollo de aplicaciones complejas y de alto rendimiento.

1.1.2. Coexistencia entre HTML, JS y CSS

El desarrollo web moderno del lado del cliente se basa en la integración y coexistencia de tres tecnologías fundamentales: HTML, CSS y JavaScript. Cada una de estas tecnologías tiene un propósito específico y se complementan entre sí para crear experiencias web dinámicas y atractivas.

HTML

HTML[2] (HyperText Markup Language) es el lenguaje estándar para la creación de documentos que se van a visualizar en un navegador web. Su propósito principal es estructurar el contenido, definiendo elementos como títulos, párrafos, listas, enlaces, imágenes y otros componentes multimedia. HTML utiliza una serie de etiquetas (*tags*) para delimitar y organizar estos elementos en una página web.

CSS

CSS[1] (Cascading Style Sheets) es un lenguaje utilizado para describir la presentación de un documento escrito en HTML o XML. Mientras que HTML se encarga de la estructura del contenido, CSS se utiliza para estilizarlo, permitiendo especificar colores, fuentes, márgenes, bordes, espaciado, y disposición de los elementos. CSS facilita la separación del contenido y la presentación, lo que permite una mayor flexibilidad y control en el diseño visual de las páginas web.

JavaScript

JavaScript[3] es un lenguaje de programación que se ejecuta en el navegador del usuario. Su propósito es hacer que las páginas web sean interactivas y dinámicas. Con JavaScript, es posible responder a eventos del usuario (como clics y movimientos del ratón), manipular el DOM (Document Object Model) de la página para modificar su contenido y estructura, y comunicarse con servidores para cargar datos adicionales sin necesidad de recargar la página completa.

Coexistencia y Colaboración

La combinación de HTML, CSS y JavaScript permite crear aplicaciones web complejas y ricas en funcionalidades:

- **HTML** proporciona la estructura básica del contenido.
- **CSS** define el estilo y la presentación visual de ese contenido.
- **JavaScript** añade interactividad y comportamiento dinámico.

  Cada uno de estos lenguajes interactúa de manera armoniosa para construir una experiencia de usuario coherente. Por ejemplo, se puede utilizar HTML para crear un formulario, CSS para diseñarlo de manera atractiva, y JavaScript para validar los datos ingresados por el usuario antes de enviarlos a un servidor.

  En resumen, mientras que HTML y CSS se enfocan en la estructura y apariencia, JavaScript se encarga de la lógica y la interacción, logrando así una sinergia que permite el desarrollo de sitios web modernos y responsivos.

1.1.3. Características del lenguaje

Algunas características clave de JavaScript son:

- **Dinámicamente tipado:** Los tipos de las variables se determinan en tiempo de ejecución y pueden cambiar durante la ejecución del programa.
- **Multiparadigma:** Soporta programación funcional y orientada a objetos.
- **Ejecución en el cliente:** Se ejecuta principalmente en el navegador del cliente, permitiendo manipulación del DOM y interactividad sin necesidad de recargar la página.
- **Asincronía:** Permite operaciones asíncronas, esenciales para la interacción con servidores y APIs.
- **Versatilidad:** Además del desarrollo web, se utiliza en desarrollo de servidores (Node.js), aplicaciones móviles (React Native), y aplicaciones de escritorio (Electron).
- **Amplio ecosistema:** Cuenta con una gran cantidad de bibliotecas y frameworks que extienden sus capacidades.

La continua evolución de ECMAScript asegura que JavaScript siga siendo un lenguaje moderno y potente, adaptándose a las necesidades cambiantes del desarrollo de software y web. Debe mencionarse también que JS es un lenguaje controvertido, con reglas particulares que pueden llevar a comportamientos no deseados de nuestras aplicaciones. Es por esto que se recomienda su uso siguiendo buenas prácticas como veremos mas adelante en repetidas ocasiones.

1.1.4. Hola Mundo en JavaScript

Para nuestro ejemplo de "Hola Mundo", crearemos una página HTML simple con un script de JavaScript que mostrará una alerta cuando la página se cargue.

Primero, veamos el código HTML:

Bloque de código 1.1: Hola mundo

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hola Mundo en JavaScript</title>
</head>
<body>
    <h1>Mi primera pagina con JavaScript</h1>
    <script>
        // Este script se ejecutara cuando la pagina se cargue
        window.onload = function() {
            alert("Hola Mundo!");
        };
    </script>
</body>
</html>
```

En este ejemplo:

- Creamos una estructura HTML básica.
- Incluimos un elemento `<script>` dentro del `<body>`.
- Usamos `window.onload` para asegurarnos de que el script se ejecute cuando la página haya terminado de cargarse.
- La función `alert()` muestra un cuadro de diálogo con el mensaje "Hola Mundo".

Observese la estructura del programa que escrito dentro del tag `<script>`. Por un lado hemos creado instrucciones. Cada instrucción debe acabar con un punto y coma `;`. Si bien el compilador de JavaScript autocompleta el punto y coma resulta una buena práctica colocarlo al final de cada instrucción.

Nótese que hemos llamado al objeto global window, mas adelante veremos algunso objetos globales. Hemos invocado la propiedad onload usando un punto seguido del nombre de esta propiedad. Mas adelante veremos objetos, propiedades y métodos.

La propiedad onload se utiliza para registrar una función que se ejecutará cuando la página se cargue. Con lo cual le hemos asignado una función. Mas adelante veremos la sintaxis de las funciones.

Dentro de la función que asignamos a onload, hemos llamado a la función global `alert()` para mostrar un cuadro de diálogo. A la función le hemos pasado como argumento un «Hola Mundo!».

Para ver este ejemplo en acción:

1. Crear un nuevo archivo y nombrarlo `index.html`.
2. Copiar y pegar el código HTML anterior en este archivo.
3. Guardar el archivo y abrirlo en un navegador web.

Cargar la página disparará un cuadro de diálogo de alerta que dirá "Hola Mundo".

Alternativamente, también se puede usar `console.log()` para mostrar mensajes en la consola del navegador:

Bloque de código 1.2: Hola mundo con console.log

```
console.log("Hola Mundo!");
```

Para ver este mensaje se debe abrir las herramientas de desarrollador del navegador (generalmente presionando F12) y seleccionar la pestaña Çonsole".

1.2. Sintaxis básica y variables

1.2.1. Declaración de variables

JavaScript tiene tres formas de declarar variables: `var`, `let`, y `const`. Sin embargo, en la programación moderna de JavaScript, se desaconseja el uso de `var`.

Bloque de código 1.3: Variables

```
// let (block-scoped, no puede ser redeclarada en el mismo scope)
let x = 5;

// const (block-scoped, no puede ser reasignada)
const PI = 3.14159;

// var (funcion-scoped, puede ser redeclarada) - NO RECOMENDADO
var y = 10;
```

Es importante entender por qué `var` no se recomienda:

- **Scope**: `var` tiene un alcance de función o global, lo que puede llevar a comportamientos inesperados y errores difíciles de detectar.
- **Hoisting**: Las variables declaradas con `var` se «elevan» al principio de su scope, lo que puede causar confusión.
- **Redeclaración**: `var` permite redeclarar la misma variable en el mismo scope sin error, lo que puede llevar a bugs sutiles.

En su lugar, se recomienda usar:

- `let` para variables que necesitan ser reasignadas.
- `const` para variables que no serán reasignadas (lo cual es preferible cuando sea posible).

Ejemplo de uso recomendado:

Bloque de código 1.4: Uso de let y const

```
let contador = 0;
contador++; // Valido, contador puede ser reasignado

const URL_API = "https://api.ejemplo.com";
// URL_API = "https://otra-api.com"; // Esto daria un error

// Uso de const con objetos
const usuario = { nombre: "Juan" };
usuario.nombre = "Maria"; // Valido, el objeto puede ser mutado
// usuario = { nombre: "Pedro" }; // Esto daria un error
```

Al usar `let` y `const` el código es más predecible, más fácil de entender y menos propenso a errores.

Es importante no asignar valores a variables no declaradas ya que esto puede volver el código confuso. JavaScript crea aquellas variables que no declaramos pero que de todas formas asignamos. Si bien JavaScript nos ofrece esta característica es mas claro no usarla y declarar todas las variables.

1.2.2. Tipos de datos

JavaScript tiene varios tipos de datos primitivos y también posee el tipo objeto, que veremos mas adelante. Los tipos de datos primitivos de JS son:

- number: enteros y decimales
- string: cadenas de caracteres
- boolean: booleanos (verdadero o falso)
- undefined: valor nulo por defecto
- null: valor nulo manual
- symbol: valor único
- BigInt: enteros muy grandes

Ejemplo:

Bloque de código 1.5: Tipos primitivos

```
let num = 42;         // number
let str = 'Hola';     // string asignado con un literal entre '
let str2 = "Hola";    // string asignado con un literal entre "
let bool = true;      // boolean
let undef = undefined;// undefined
let nul = null;       // null
let sym = Symbol();   // symbol
let big = 12345678901234567890123456789012345678901234567890n; // BigInt
```

En el ejemplo puede observarse que los literales de string pueden crearse con comillas simples o dobles. En general usaremos comillas simples.

Ademas de las comillas simples los strings pueden crearse con comillas invertidas (‘). Este tipo de cadenas permite la interpolación, una forma mas cómoda de insertar valores en un string que fue introducida en ECMAScript 6. Veamos un ejemplo:

Bloque de código 1.6: Interpolación de cadenas

```
const nombre = 'Juan';
const mensajeDespues = ‘Hola, mi nombre es ${nombre}.‘;
```

Lo cual arrojará por consola

```
Hola, mi nombre es Juan.
```

El valor NaN

NaN (Not a Number) es un valor que representa un error de tipo numérico. Por ejemplo, el resultado de la división entre 0 es NaN.

Ejemplo:

```
let num = 42;
let div = num / 0;
console.log(div); // NaN
```

Además NaN es parte del objeto global Number, mas adelante veremos qué son estos objetos.

NaN nunca es equivalente con cualquier otro número, incluido el mismo NaN; no se puede chequear el valor de un not-a-number comparándolo con Number.NaN. Se debe usar la función isNaN() para esto.

1.2.3. Operadores

JavaScript incluye varios tipos de operadores. Veamos cada tipo con ejemplos:

Operadores Aritméticos

Estos operadores realizan operaciones matemáticas.

```
let a = 10;
let b = 3;

console.log(a + b);  // Suma: 13
console.log(a - b);  // Resta: 7
console.log(a * b);  // Multiplicacion: 30
console.log(a / b);  // Division: 3.3333...
console.log(a % b);  // Modulo (resto): 1
console.log(a ** b); // Exponenciacion: 1000
```

Operadores de Asignación

Estos operadores asignan valores a variables.

```
let x = 5;
console.log(x);  // 5

x += 3;  // Equivalente a: x = x + 3
console.log(x);  // 8

x -= 2;  // Equivalente a: x = x - 2
console.log(x);  // 6

x *= 4;  // Equivalente a: x = x * 4
console.log(x);  // 24
```

```
x /= 3;  // Equivalente a: x = x / 3
console.log(x);  // 8

x %= 3;  // Equivalente a: x = x % 3
console.log(x);  // 2
```

Operadores de Comparación

Estos operadores comparan valores y devuelven un booleano.

```
let a = 5;
let b = '5';

console.log(a == b);   // Igualdad (con coercion): true
console.log(a === b);  // Igualdad estricta: false
console.log(a != b);   // Desigualdad (con coercion): false
console.log(a !== b);  // Desigualdad estricta: true
console.log(a > 3);    // Mayor que: true
console.log(a < 3);    // Menor que: false
console.log(a >= 5);   // Mayor o igual que: true
console.log(a <= 4);   // Menor o igual que: false
```

Operadores Lógicos

Estos operadores trabajan con valores booleanos.

```
let x = true;
let y = false;

console.log(x && y);  // AND logico: false
console.log(x || y);  // OR logico: true
console.log(!x);      // NOT logico: false

// Uso con no-booleanos
console.log(5 && 2);  // 2 (retorna el ultimo valor verdadero)
console.log(0 || 3);  // 3 (retorna el primer valor verdadero)
console.log(!'');     // true ('' es falsy)
```

Operadores Unarios

Estos operadores trabajan con un solo operando.

```
let a = 5;

console.log(a++);  // Incremento posterior: 5 (a ahora es 6)
console.log(++a);  // Incremento anterior: 7

console.log(a--);  // Decremento posterior: 7 (a ahora es 6)
console.log(--a);  // Decremento anterior: 5

console.log(+a);   // Conversion a numero: 5
console.log(-a);   // Negacion unaria: -5
```

Operador Ternario

Este operador es una forma concisa de escribir una declaración if-else.

```
let edad = 20;
let mensaje = (edad >= 18) ? "Adulto" : "Menor";
console.log(mensaje);  // "Adulto"

// Equivalente a:
// if (edad >= 18) {
//     mensaje = "Adulto";
// } else {
//     mensaje = "Menor";
// }
```

Estos ejemplos muestran cómo se utilizan los diferentes tipos de operadores en JavaScript. Es importante entender cómo funcionan para escribir código eficiente y evitar errores comunes.

1.3. Estructuras de control

Las estructuras de control en JavaScript permiten alterar el flujo de ejecución del código basándose en ciertas condiciones o repetir bloques de código un número determinado de veces.

1.3.1. Condicionales

Las estructuras condicionales permiten ejecutar diferentes bloques de código dependiendo de si una condición se cumple o no.

if-else

La estructura `if-else` permite ejecutar un bloque de código si una condición es verdadera, y otro bloque si es falsa.

```
let edad = 18;

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

También se puede usar `else if` para comprobar múltiples condiciones:

```
let nota = 75;

if (nota >= 90) {
    console.log("Sobresaliente");
} else if (nota >= 70) {
    console.log("Notable");
} else if (nota >= 50) {
    console.log("Aprobado");
} else {
    console.log("Suspenso");
}
```

switch

La estructura `switch` se utiliza cuando se quiere comparar una variable con múltiples valores posibles. Es una alternativa más legible a múltiples `if-else` cuando se compara una sola variable.

```
let dia = "Lunes";

switch (dia) {
    case "Lunes":
        console.log("Inicio de semana");
        break;
    case "Viernes":
        console.log("Fin de semana laboral");
        break;
    case "Sabado":
    case "Domingo":
        console.log("Fin de semana");
        break;
    default:
        console.log("Mitad de semana");
}
```

1.3.2. Bucles

Los bucles permiten ejecutar un bloque de código repetidamente mientras se cumpla una condición.

for

El bucle `for` se utiliza cuando se conoce de antemano el número de iteraciones que se quieren realizar. Consta de tres partes: inicialización, condición y expresión final.

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

while

El bucle `while` se ejecuta mientras una condición sea verdadera. Es útil cuando no se sabe de antemano cuántas iteraciones se necesitarán.

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

do-while

Similar al bucle `while`, pero garantiza que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa al final de cada iteración.

```
let i = 0;
do {
    console.log(i);
    i++;
} while (i < 5);
```

1.3.3. Otras estructuras de control

break y continue

`break` se utiliza para salir inmediatamente de un bucle, mientras que `continue` salta a la siguiente iteración del bucle.

```
for (let i = 0; i < 5; i++) {
    if (i === 2) continue; // Salta esta iteracion
    if (i === 4) break;    // Sale del bucle
    console.log(i);
}
// Salida: 0, 1, 3
```

Estas estructuras de control son fundamentales en JavaScript y en la programación en general, ya que permiten crear lógica compleja y controlar el flujo de ejecución del programa.

1.4. Funciones en JavaScript

Las funciones son bloques de código reutilizables que realizan una tarea específica. Son fundamentales en JavaScript y permiten estructurar el código de manera modular y eficiente.

1.4.1. Declaración de Funciones

Hay varias formas de declarar funciones en JavaScript:

Declaración de Función

```
function saludar(nombre) {
    console.log("Hola, " + nombre + "!");
}

saludar("Maria"); // Salida: Hola, Maria!
```

Expresión de Función

```
const saludar = function(nombre) {
    console.log("Hola, " + nombre + "!");
};

saludar("Juan"); // Salida: Hola, Juan!
```

Función Flecha (Arrow Function)

Introducida en ES6, ofrece una sintaxis más concisa:

```
const saludar = (nombre) => {
    console.log("Hola, " + nombre + "!");
};

saludar("Ana"); // Salida: Hola, Ana!
```

Para funciones de una sola línea, se puede simplificar aún más:

```
const cuadrado = x => x * x;

console.log(cuadrado(4)); // Salida: 16
```

1.4.2. Parámetros y Argumentos

Los parámetros son variables listadas como parte de la definición de la función. Los argumentos son los valores reales pasados a la función cuando se llama.

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

console.log(sumar(3, 4)); // Salida: 7
```

Parámetros por Defecto

Se pueden asignar valores por defecto a los parámetros:

```
function saludar(nombre = "Invitado") {
    console.log("Hola, " + nombre + "!");
}

saludar(); // Salida: Hola, Invitado!
saludar("Pedro"); // Salida: Hola, Pedro!
```

El problema con el tipado débil

Cómo hemos visto, la sintaxis de las funciones de JavaScript es más flexible que la de otros lenguajes. No se especifica ni el tipo que retorna la función ni el tipo de los parámetros. Esto puede ser una ventaja en algunas ocasiones. Pero también puede resultar problemático. No saber que tipo de dato debemos proveer a una función o que tipo de dato nos retorna implica que no podemos usarla hasta tanto no conozcamos de alguna manera esta información. Por eso es muy importante el uso de comentarios y documentación, de lo contrario nuestro código puede volverse en sumo desastroso o incomprensible muy rápidamente.

1.4.3. Retorno de Valores

Las funciones pueden devolver valores usando la palabra clave `return`:

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

let resultado = multiplicar(4, 5);
console.log(resultado); // Salida: 20
```

Si no se especifica un valor de retorno, la función devolverá `undefined`.

1.4.4. Scope y Closures

El scope de una función define la accesibilidad de las variables:

```
let global = "Soy global";

function mostrarVariables() {
    let local = "Soy local";
    console.log(global); // Accesible
    console.log(local);  // Accesible
}

mostrarVariables();
// console.log(local); // Error: local no esta definida
```

Un closure es una función que tiene acceso a variables en su scope externo incluso después de que la función externa haya retornado:

```
function crearContador() {
    let contador = 0;
    return function() {
        return ++contador;
    };
}

let contar = crearContador();
console.log(contar()); // Salida: 1
console.log(contar()); // Salida: 2
```

1.4.5. Funciones Inmediatamente Invocadas (IIFE)

Son funciones que se ejecutan tan pronto como se definen:

```
(function() {
    let mensaje = "Hola desde IIFE";
    console.log(mensaje);
})();
```

Las funciones son una parte crucial de JavaScript, permitiendo escribir código modular, reutilizable y bien organizado. Dominar el uso de funciones es esencial para cualquier desarrollador de JavaScript.

Capítulo 2

Objetos y manipulación del DOM

2.1. Objetos en JavaScript y Paradigmas de Programación

2.1.1. Objetos en JavaScript

Los objetos en JavaScript son estructuras de datos que permiten almacenar colecciones de pares clave-valor. Son fundamentales en el lenguaje y se utilizan para representar entidades del mundo real o conceptos abstractos.

Propiedades y Métodos

- **Propiedades:** Son las características o atributos del objeto. Se representan como pares clave-valor, donde la clave es el nombre de la propiedad y el valor puede ser cualquier tipo de dato válido en JavaScript.
- **Métodos:** Son funciones asociadas a un objeto. Representan las acciones que el objeto puede realizar.

Creación de Objetos

```
// Usando la notacion literal
let persona = {
    nombre: "Ana",     // propiedad
    edad: 30,          // propiedad
    saludar: function() {  // metodo
        console.log("Hola, soy " + this.nombre);
    }
};

// Usando el constructor Object()
let coche = new Object();
coche.marca = "Toyota";  // aniadiendo propiedades
coche.modelo = "Corolla";
coche.anio = 2022;
```

Acceso a Propiedades y Métodos

```
console.log(persona.nombre);  // Acceso a propiedad: Ana
console.log(coche["modelo"]); // Acceso alternativo: Corolla
persona.saludar();            // Llamada a metodo: Hola, soy Ana
```

Modificación y Adición de Propiedades

```
persona.edad = 31;             // Modificacion de propiedad existente
persona.profesion = "Ingeniera"; // Adicion de nueva propiedad
```

2.1.2. Funciones como Objetos de Primera Clase

En JavaScript, las funciones son objetos de primera clase, lo que significa que pueden:

- Asignarse a variables
- Pasarse como argumentos a otras funciones

- Devolverse como valores de otras funciones

```
// Funcion como argumento
function ejecutar(fn, valor) {
    return fn(valor);
}

let duplicar = x => x * 2;
console.log(ejecutar(duplicar, 5)); // Salida: 10

// Funcion que devuelve otra funcion
function crearMultiplicador(factor) {
    return function(x) {
        return x * factor;
    };
}

let duplicar = crearMultiplicador(2);
console.log(duplicar(4)); // Salida: 8
```

2.1.3. Programación Orientada a Objetos (POO) en JavaScript

La POO es un paradigma de programación que organiza el diseño de software en torno a los objetos, en lugar de funciones y lógica. JavaScript, aunque es un lenguaje multiparadigma, soporta la POO. Aunque debe advertirse que los objetos en este lenguaje pueden ser considerados como mapas para quienes vengan de otros lenguajes de programación.

Clases en JavaScript (ES6+)

Una clase es un plano para crear objetos. Define las propiedades y métodos que tendrán los objetos creados a partir de ella.

```
class Persona {
    constructor(nombre, edad) {
        this.nombre = nombre; // Inicializacion de propiedades
        this.edad = edad;
    }

    saludar() { // Metodo de la clase
        console.log(‘Hola, soy ${this.nombre} y tengo ${this.edad} anios.‘);
    }
}

let ana = new Persona("Ana", 30); // Creacion de un objeto
ana.saludar(); // Hola, soy Ana y tengo 30 anios.
```

Herencia

La herencia es un mecanismo que permite que una clase (clase hija) herede propiedades y métodos de otra clase (clase padre). Esto promueve la reutilización de código y la creación de jerarquías de objetos.

```
class Empleado extends Persona {
    constructor(nombre, edad, puesto) {
        super(nombre, edad); // Llama al constructor de la clase padre
        this.puesto = puesto; // Propiedad especifica de Empleado
    }

    trabajar() { // Metodo especifico de Empleado
        console.log(‘${this.nombre} esta trabajando como ${this.puesto}.‘);
    }
}

let juan = new Empleado("Juan", 25, "desarrollador");
juan.saludar();  // Metodo heredado: Hola, soy Juan y tengo 25 anios.
juan.trabajar(); // Metodo propio: Juan esta trabajando como desarrollador.
```

En este ejemplo, `Empleado` hereda de `Persona`, lo que significa que un `Empleado` tiene todas las propiedades y métodos de `Persona`, más sus propias propiedades y métodos adicionales.

La POO en JavaScript permite organizar el código de manera más estructurada y reutilizable, facilitando la creación de aplicaciones complejas y mantenibles.

2.1.4. Otro paradigma: la Programación Funcional

Además de la POO, JavaScript también soporta el paradigma de programación funcional. Este enfoque se centra en el uso de funciones para resolver problemas, tratando la computación como la evaluación de funciones matemáticas y evitando el cambio de estado y datos mutables. JavaScript, siendo un lenguaje multiparadigma, permite a los desarrolladores utilizar técnicas de programación funcional.

2.2. El DOM y Manipulación de Eventos

2.2.1. ¿Qué es el DOM?

El DOM (Document Object Model) es una representación estructurada del documento HTML como un árbol de objetos. Permite a JavaScript acceder y manipular el contenido, estructura y estilo de una página web.

```
// Ejemplo de estructura del DOM
// <html>
//   <head>
//     <title>Mi Pagina</title>
//   </head>
//   <body>
//     <h1>Bienvenido</h1>
//     <p>Este es un parrafo.</p>
//   </body>
// </html>
```

2.2.2. Manipulación del DOM

JavaScript puede modificar todos los elementos y atributos HTML, así como los estilos CSS en una página.

Selección de Elementos

```
// Por ID
let elemento = document.getElementById('miId');

// Por clase
let elementos = document.getElementsByClassName('miClase');

// Por etiqueta
let parrafos = document.getElementsByTagName('p');

// Usando selectores CSS
let primerElemento = document.querySelector('.miClase');
let todosElementos = document.querySelectorAll('.miClase');
```

Modificación de Elementos

```
// Cambiar contenido
elemento.textContent = 'Nuevo texto';
elemento.innerHTML = '<strong>Texto en negrita</strong>';

// Cambiar atributos
elemento.setAttribute('class', 'nuevaClase');

// Cambiar estilos
elemento.style.color = 'red';
elemento.style.fontSize = '20px';

// Agregar/Eliminar clases
elemento.classList.add('nuevaClase');
elemento.classList.remove('viejaClase');
```

Creación y Eliminación de Elementos

```
// Crear nuevo elemento
let nuevoElemento = document.createElement('div');
nuevoElemento.textContent = 'Nuevo elemento';

// Agregar al DOM
document.body.appendChild(nuevoElemento);

// Eliminar elemento
let elementoAEliminar = document.getElementById('eliminar');
elementoAEliminar.parentNode.removeChild(elementoAEliminar);
```

2.2.3. El Elemento Button

El elemento `<button>` en HTML crea un botón interactivo en la página web.

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

En JavaScript, puedes interactuar con el botón:

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

// Cambiar texto del boton
boton.textContent = 'Nuevo texto';

// Deshabilitar/habilitar el boton
boton.disabled = true;
boton.disabled = false;

// Cambiar estilos
boton.style.backgroundColor = 'blue';
boton.style.color = 'white';
```

2.2.4. Escuchadores de Eventos

Los escuchadores de eventos permiten que JavaScript reaccione a acciones del usuario o del navegador.

Agregar un Escuchador de Eventos

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

boton.addEventListener('click', function() {
    console.log('Boton clickeado');
});
```

Tipos Comunes de Eventos

- `click`: Cuando se hace clic en un elemento.
- `mouseover` / `mouseout`: Cuando el cursor entra o sale de un elemento.
- `keydown` / `keyup`: Cuando se presiona o suelta una tecla.
- `submit`: Cuando se envía un formulario.
- `load`: Cuando la página ha terminado de cargar.

Ejemplo Práctico

```
<button id="cambiarColor">Cambiar Color</button>
<div id="cuadrado" style="width:100px;height:100px;background-color:red;"></div>
```

```
let boton = document.getElementById('cambiarColor');
let cuadrado = document.getElementById('cuadrado');

boton.addEventListener('click', function() {
    let nuevoColor = cuadrado.style.backgroundColor === 'red' ? 'blue' : 'red';
    cuadrado.style.backgroundColor = nuevoColor;
});
```

2.2.5. Propagación de Eventos y Prevención del Comportamiento por Defecto

Propagación de Eventos (Event Bubbling)

Los eventos en el DOM se propagan desde el elemento más interno hacia afuera.

```
document.body.addEventListener('click', function() {
    console.log('Clic en el body');
}, false);

boton.addEventListener('click', function(event) {
    console.log('Clic en el boton');
    event.stopPropagation(); // Detiene la propagacion
}, false);
```

Prevenir el Comportamiento por Defecto

```
let enlace = document.getElementById('miEnlace');

enlace.addEventListener('click', function(event) {
    event.preventDefault(); // Previene la navegacion
    console.log('Clic en el enlace');
});
```

La manipulación del DOM y el manejo de eventos son fundamentales para crear páginas web interactivas. Permiten a los desarrolladores responder a las acciones del usuario y modificar dinámicamente el contenido y la apariencia de la página.

2.3. Ejemplo Integrador: Contador Interactivo

Para poner en práctica lo que hemos aprendido hasta ahora, vamos a crear un ejemplo integrador que combine HTML y JavaScript. Crearemos una página web simple con un contador que el usuario puede incrementar, decrementar y resetear usando botones.

2.3.1. Estructura HTML

Primero, creemos la estructura HTML básica con los elementos necesarios:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Contador Interactivo</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        button {
            font-size: 18px;
            margin: 0 10px;
            padding: 5px 10px;
        }
        #contador {
            font-size: 24px;
            margin: 20px 0;
        }
    </style>
</head>
<body>
    <h1>Contador Interactivo</h1>
    <div id="contador">0</div>
    <button id="decrementar">-</button>
    <button id="incrementar">+</button>
    <button id="resetear">Resetear</button>

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

2.3.2. JavaScript (contador.js)

Ahora, creemos el archivo JavaScript que manejará la lógica del contador:

```
// Variables
let contador = 0;
const valorContador = document.getElementById('contador');
const btnDecrementar = document.getElementById('decrementar');
const btnIncrementar = document.getElementById('incrementar');
const btnResetear = document.getElementById('resetear');

// Funciones
function actualizarContador() {
    valorContador.textContent = contador;

    // Cambia el color basado en el valor
    if (contador > 0) {
        valorContador.style.color = 'green';
    } else if (contador < 0) {
```

```
        valorContador.style.color = 'red';
    } else {
        valorContador.style.color = 'black';
    }
}

function incrementar() {
    contador++;
    actualizarContador();
}

function decrementar() {
    contador--;
    actualizarContador();
}

function resetear() {
    contador = 0;
    actualizarContador();
}

// Event Listeners
btnIncrementar.addEventListener('click', incrementar);
btnDecrementar.addEventListener('click', decrementar);
btnResetear.addEventListener('click', resetear);

// Inicializar el contador
actualizarContador();
```

2.3.3. Explicación del Código

- Usamos `document.getElementById()` para obtener referencias a los elementos HTML.
- Definimos funciones para incrementar, decrementar y resetear el contador.
- La función `actualizarContador()` actualiza el valor mostrado y cambia el color basado en el valor actual.
- Utilizamos `addEventListener()` para asociar las funciones a los eventos de clic de los botones.
- El color del contador cambia a verde si es positivo, rojo si es negativo, y negro si es cero.

2.3.4. Conceptos Aplicados

Este ejemplo integra varios conceptos que hemos aprendido:

- Variables y constantes (`let`, `const`)
- Funciones
- Estructuras condicionales (`if-else`)
- Operadores de incremento y decremento
- Manipulación del DOM
- Event Listeners

2.3.5. Probando el Ejemplo

Para probar este ejemplo:

1. Crea un archivo HTML con el contenido proporcionado y guárdalo como `index.html`.
2. Crea un archivo JavaScript con el código proporcionado y guárdalo como `contador.js` en la misma carpeta.
3. Abre el archivo HTML en un navegador web.

Verás un contador con botones para incrementar, decrementar y resetear. El valor cambiará de color según sea positivo, negativo o cero.

Este ejemplo práctico demuestra cómo los conceptos básicos de JavaScript se pueden aplicar para crear una interfaz de usuario interactiva simple.

Capítulo 3

Estructuras de datos

3.1. Arrays

Un array es una colección ordenada de elementos. En JavaScript, los arrays pueden contener elementos de cualquier tipo de datos, incluidos otros arrays.

3.1.1. Creación de Arrays

Para crear un array en JavaScript, se pueden utilizar corchetes ‘[]‘ o el constructor ‘Array‘.

```
let array1 = [1, 2, 3, 4, 5];
let array2 = new Array(1, 2, 3, 4, 5);
```

3.1.2. Acceso a Elementos

Los elementos de un array se acceden utilizando índices, comenzando desde 0.

```
let firstElement = array1[0]; // 1
let lastElement = array1[array1.length - 1]; // 5
```

3.1.3. Métodos de Arrays

JavaScript proporciona varios métodos útiles para manipular arrays, tales como ‘push‘, ‘pop‘, ‘shift‘, ‘unshift‘, ‘map‘, ‘filter‘, y ‘reduce‘.

```
array1.push(6); // Agrega 6 al final del array
array1.pop(); // Remueve el ultimo elemento del array
let newArray = array1.map(x => x * 2); // [2, 4, 6, 8, 10]
let filteredArray = array1.filter(x => x > 2); // [3, 4, 5]
let sum = array1.reduce((acc, x) => acc + x, 0); // 15
```

3.1.4. Rest Parameters en funciones

Ahora que estamos mejor familiarizados con los arrays en JavaScript, vamos a ver un ejemplo de uso de rest parameters en una función. Este tipo de parámetros permiten representar un número indefinido de argumentos como un array:

```
function sumar(...numeros) {
    return numeros.reduce((total, num) => total + num, 0);
}

console.log(sumar(1, 2, 3, 4)); // Salida: 10
```

3.2. Sets

Un set es una colección de valores únicos. Los sets permiten almacenar cualquier tipo de valor, ya sea primitivo u objeto.

3.2.1. Creación de Sets

Para crear un set en JavaScript, se utiliza el constructor ‘Set‘.

```
let set = new Set([1, 2, 3, 4, 5, 5]);
console.log(set); // Set { 1, 2, 3, 4, 5 }
```

3.2.2. Métodos de Sets

Los sets tienen métodos para añadir, eliminar y verificar la existencia de elementos.

```
set.add(6); // Agrega 6 al set
set.delete(3); // Elimina 3 del set
console.log(set.has(4)); // true
```

3.3. Maps

Un map es una colección de pares clave-valor donde las claves pueden ser de cualquier tipo de datos.

3.3.1. Creación de Maps

Para crear un map en JavaScript, se utiliza el constructor ‘Map‘.

```
let map = new Map();
map.set("name", "Dave");
map.set("age", 40);
```

3.3.2. Métodos de Maps

Los maps tienen métodos para añadir, eliminar y obtener elementos.

```
console.log(map.get("name")); // "Dave"
map.delete("age"); // Elimina la clave "age"
console.log(map.has("age")); // false
```

3.4. Ejemplo Integrador

A continuación, presentaremos un ejemplo que integra un array con HTML para crear un menú desplegable (‘<select>‘) y un botón que muestra una alerta con la opción seleccionada.

3.4.1. Código HTML y JavaScript

El siguiente código HTML incluye un menú desplegable y un botón. El JavaScript asociado llena el menú desplegable con valores de un array y muestra una alerta con la opción seleccionada cuando se hace clic en el botón.

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo Integrador</title>
    <script type="text/javascript">
        document.addEventListener('DOMContentLoaded', function() {
            let optionsArray = ['Opcion 1', 'Opcion 2', 'Opcion 3', 'Opcion 4'];
            let selectElement = document.getElementById('optionsSelect');
            optionsArray.forEach(function(option) {
                let optionElement = document.createElement('option');
                optionElement.textContent = option;
                optionElement.value = option;
                selectElement.appendChild(optionElement);
            });

            document.getElementById('alertButton').addEventListener('click', function() {
                let selectedOption = selectElement.value;
                alert('Has seleccionado: ' + selectedOption);
            });
        });
    </script>
</head>
<body>
    <h1>Selecciona una Opcion</h1>
    <select id="optionsSelect"></select>
    <button id="alertButton">Mostrar Seleccion</button>
</body>
</html>
```

3.4.2. Explicación del Código

El script JavaScript se ejecuta cuando el contenido del DOM ha sido completamente cargado. Se define un array de opciones y se utiliza para llenar el elemento ‘<select>‘ con opciones. Al hacer clic en el botón, se muestra una alerta con la opción seleccionada.

- **‘optionsArray‘**: Array que contiene las opciones del menú desplegable.
- **‘selectElement‘**: Referencia al elemento ‘<select>‘ en el DOM.
- **‘forEach‘**: Itera sobre cada elemento del array, creando y añadiendo un elemento ‘<option>‘ al menú desplegable.
- **‘addEventListener‘**: Añade un evento de clic al botón que muestra una alerta con la opción seleccionada.

Capítulo 4

Errores, Objetos Globales, Funciones Globales y asincronía en JavaScript

4.1. Manejo de Errores

JavaScript proporciona mecanismos para manejar errores y excepciones que pueden ocurrir durante la ejecución del código.

4.1.1. Try...Catch

El bloque try...catch es la estructura básica para manejar errores en JavaScript:

```
try {
  // Codigo que puede lanzar un error
  throw new Error('Este es un error de ejemplo');
} catch (error) {
  console.error('Se capturo un error:', error.message);
} finally {
  console.log('Este bloque siempre se ejecuta');
}
```

4.1.2. Tipos de Errores

JavaScript tiene varios tipos de errores predefinidos:

- `Error`: Error genérico
- `SyntaxError`: Error de sintaxis
- `ReferenceError`: Referencia a una variable no definida
- `TypeError`: Operación en un tipo de dato incorrecto
- `RangeError`: Valor numérico fuera de rango

4.1.3. Creación de Errores Personalizados

Se pueden crear errores personalizados extendiendo la clase Error:

```
class MiErrorPersonalizado extends Error {
  constructor(mensaje) {
    super(mensaje);
    this.name = 'MiErrorPersonalizado';
  }
}

throw new MiErrorPersonalizado('Este es un error personalizado');
```

4.2. Objetos Globales Populares

JavaScript tiene varios objetos globales incorporados que proporcionan funcionalidad útil.

4.2.1. Object

El objeto raíz de la jerarquía de objetos en JavaScript:

```
const obj = { a: 1, b: 2 };
console.log(Object.keys(obj)); // ['a', 'b']
console.log(Object.values(obj)); // [1, 2]
```

4.2.2. Array

Proporciona métodos para trabajar con arreglos:

```
const arr = [1, 2, 3];
console.log(arr.map(x => x * 2)); // [2, 4, 6]
console.log(arr.filter(x => x > 1)); // [2, 3]
```

4.2.3. String

Ofrece métodos para manipular cadenas de texto:

```
const str = 'Hello, World!';
console.log(str.toLowerCase()); // 'hello, world!'
console.log(str.split(', ')); // ['Hello', 'World!']
```

4.2.4. Math

Proporciona funciones matemáticas y constantes:

```
console.log(Math.PI); // 3.141592653589793
console.log(Math.random()); // Numero aleatorio entre 0 y 1
```

4.2.5. Date

Permite trabajar con fechas y horas:

```
const ahora = new Date();
console.log(ahora.toISOString());
```

4.3. Funciones Globales Populares

JavaScript incluye varias funciones globales que pueden ser utilizadas sin necesidad de un objeto.

4.3.1. parseInt() y parseFloat()

Convierten cadenas a números:

```
console.log(parseInt('42')); // 42
console.log(parseFloat('3.14')); // 3.14
```

4.3.2. isNaN() y isFinite()

Verifican si un valor es NaN (Not a Number) o finito:

```
console.log(isNaN(NaN)); // true
console.log(isFinite(1/0)); // false
```

4.3.3. encodeURIComponent() y decodeURIComponent()

Codifican y decodifican componentes de URI:

```
const codificado = encodeURIComponent('Hello World!');
console.log(codificado); // 'Hello%20World!'
console.log(decodeURIComponent(codificado)); // 'Hello World!'
```

4.3.4. setTimeout() y setInterval()

Permiten ejecutar código después de un retraso o a intervalos regulares:

```
setTimeout(() => console.log('Retrasado'), 1000);

const intervalo = setInterval(() => console.log('Intervalo'), 1000);
// Para detener el intervalo:
// clearInterval(intervalo);
```

Estas funciones y objetos globales forman una parte fundamental de JavaScript, proporcionando herramientas esenciales para el manejo de errores, manipulación de datos y control del flujo de ejecución.

4.4. Asincronía clásica: Callbacks

JavaScript es un lenguaje de programación de un solo hilo. Si el único hilo de ejecución que poseemos se queda ejecutando una operación muy larga estamos en un problema. Para sortear esto es que el lenguaje permite operaciones asíncronas.

Una operación asíncrona es una operación o sección del código que se especifica que debe ejecutarse en algún momento, pero no se especifica cuando, de ahí su nombre .ªsíncrona".

Una vez que el interprete de JavaScript alcansa una sección de código asíncrona deja esta porción de código en espera y gestiona según un criterio que depende del intérprete el momento en el que se ejecutará. Esto le da flexibilidad al interprete para evitar que operaciones de mucho costo computacional o de mucha latencia se ejecuten y no permitan que la interfaz sea fluida. Esto permite optimizar el uso del único hilo que del cual dispone javascript.

Para especificar que una porción de código es asíncrona se utilizan callbacks y promesas, además de un nuevo enfoque conocido como await async.

Un callback es una función que se pasa como argumento a otra función y se ejecuta después de que esta última haya terminado.

```
function operacionAsincrona(callback) {
  setTimeout(() => {
    console.log("Operacion completada");
    callback();
  }, 1000);
}

operacionAsincrona(() => {
  console.log("Callback ejecutado");
});
```

En este ejemplo, `setTimeout` simula una operación asíncrona que tarda 1 segundo en completarse.

4.5. Promesas

Las promesas proporcionan una forma más elegante de manejar operaciones asíncronas. Una promesa representa un valor que puede no estar disponible inmediatamente, pero lo estará en el futuro.

```
function operacionPromesa() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log("Operacion completada");
      resolve("Exito");
    }, 1000);
  });
}

operacionPromesa()
  .then(resultado => console.log(resultado))
  .catch(error => console.error(error));
```

Las promesas tienen tres estados: pendiente, cumplida o rechazada. Utilizamos `.then()` para manejar el éxito y `.catch()` para manejar errores.

4.6. Async/Await

Async/Await es una sintaxis más reciente que facilita el trabajo con promesas:

```
async function ejecutarOperacion() {
  try {
    const resultado = await operacionPromesa();
    console.log(resultado);
```

```
  } catch (error) {
    console.error(error);
  }
}

ejecutarOperacion();
```

4.7. Fetch API

La API Fetch proporciona una interfaz para realizar solicitudes HTTP. Devuelve una promesa que se resuelve con la respuesta a la solicitud.

```
async function generarUUID() {
  try {
    const respuesta = await fetch('https://www.uuidtools.com/api/generate/v4');
    if (!respuesta.ok) {
      throw new Error('Error en la respuesta de la red');
    }
    const datos = await respuesta.json();
    console.log('UUID generado:', datos[0]);
  } catch (error) {
    console.error('Error al generar UUID:', error);
  }
}

generarUUID();
```

Este ejemplo utiliza `fetch` para hacer una solicitud GET a la API de UUIDTools para generar un UUID versión 4. La respuesta se procesa como JSON y se muestra el UUID generado.

La función `fetch` devuelve una promesa que se resuelve con un objeto `Response`. Utilizamos `await` dos veces: una para esperar la respuesta de la red y otra para parsear el cuerpo de la respuesta como JSON.

Este ejemplo integra los conceptos de asincronía, promesas y el uso de la API Fetch en una aplicación práctica.

Bibliografía

[1] Mozilla Developer Network. *CSS: Cascading Style Sheets*. 2005. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (visitado 28-07-2024).

[2] Mozilla Developer Network. *HTML: HyperText Markup Language*. 2005. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (visitado 28-07-2024).

[3] Mozilla Developer Network. *JavaScript*. 2005. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visitado 28-07-2024).

Apéndice A

Introducción a HTML

HTML (HyperText Markup Language) es el lenguaje estándar para crear y diseñar páginas web. Proporciona la estructura básica de una página web, que es mejorada y modificada por otras tecnologías como CSS (Cascading Style Sheets) y JavaScript.

A.1. Estructura Básica de un Documento HTML

Un documento HTML típico tiene la siguiente estructura básica:

```
<!DOCTYPE html>
<html>
<head>
    <title>Titulo de la Pagina</title>
</head>
<body>
    <h1>Encabezado Principal</h1>
    <p>Este es un parrafo de texto.</p>
</body>
</html>
```

- `<!DOCTYPE html>`: Declara el tipo de documento y la versión de HTML que se está utilizando.
- `<html>`: El elemento raíz que contiene todo el contenido de la página.
- `<head>`: Contiene meta-información sobre el documento, como el título y enlaces a hojas de estilo.
- `<title>`: Establece el título de la página que aparece en la pestaña del navegador.
- `<body>`: Contiene el contenido visible de la página web, como texto, imágenes, enlaces, etc.
- `<h1>`: Define un encabezado de nivel 1, utilizado para títulos importantes.
- `<p>`: Define un párrafo de texto.

A.2. Elementos Comunes de HTML

HTML utiliza una serie de elementos (o etiquetas) para definir diferentes tipos de contenido y su estructura dentro de la página. Algunos de los elementos más comunes incluyen:

- `<h1>... <h6>`: Encabezados de diferentes niveles.
- `<p>`: Párrafos de texto.
- `<a>`: Enlaces a otras páginas o recursos.
- `<img>`: Imágenes.
- `<ul>`, `<ol>`, `<li>`: Listas no ordenadas y ordenadas.
- `<div>`: División o sección de la página.
- `<span>`: Contenedor en línea para estilizar partes del texto.

A.3. Atributos en HTML

Los elementos HTML pueden tener atributos que proporcionan información adicional sobre el elemento. Los atributos se colocan dentro de la etiqueta de apertura y generalmente consisten en un nombre y un valor.

Ejemplo:

```
<a href="https://www.ejemplo.com">Visita Ejemplo.com</a>
<img src="imagen.jpg" alt="Descripción de la imagen">
```

En este ejemplo:

- El atributo `href` del elemento `<a>` especifica la URL a la que apunta el enlace.
- El atributo `src` del elemento `<img>` especifica la ruta de la imagen.
- El atributo `alt` proporciona un texto alternativo para la imagen, que es útil para accesibilidad.

HTML es el cimiento sobre el cual se construyen las páginas web. Al proporcionar la estructura básica y semántica del contenido, HTML permite que los navegadores web muestren información de manera organizada y accesible. Aunque HTML por sí solo no define el estilo visual ni el comportamiento interactivo de una página web, funciona en conjunto con CSS y JavaScript para crear experiencias web completas y dinámicas.

Apéndice B

Introducción a CSS

CSS (Cascading Style Sheets) es un lenguaje utilizado para describir la presentación de un documento escrito en HTML o XML. Con CSS, puedes controlar el diseño y la apariencia de múltiples páginas web al mismo tiempo.

B.1. Sintaxis Básica

La sintaxis de CSS está compuesta por selectores y declaraciones. Un selector apunta al elemento HTML que quieres estilizar. Una declaración está compuesta por una propiedad y su valor.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            background-color: lightblue;
        }

        h1 {
            color: white;
            text-align: center;
        }

        p {
            font-family: verdana;
            font-size: 20px;
        }
    </style>
</head>
<body>

<h1>Este es un encabezado</h1>
<p>Este es un parrafo.</p>

</body>
</html>
```

B.1.1. Selectores

Los selectores en CSS son usados para seleccionar los elementos HTML que deseas estilizar. Algunos ejemplos de selectores son:

- Selector de elemento: selecciona elementos HTML por su nombre. Ejemplo: `h1`, `p`.
- Selector de clase: selecciona elementos con un atributo de clase específico. Se usa un punto (`.`) seguido del nombre de la clase. Ejemplo: `.clase`.
- Selector de ID: selecciona un elemento con un atributo de ID específico. Se usa una almohadilla (`#`) seguida del ID del elemento. Ejemplo: `#id`.

B.1.2. Propiedades y Valores

Las propiedades son aspectos del diseño que deseas cambiar, como el color, el tamaño de la fuente o el margen. Los valores son las configuraciones que deseas aplicar a esas propiedades.

- `color`: establece el color del texto.
- `font-size`: establece el tamaño de la fuente.
- `margin`: establece el margen alrededor del elemento.

B.2. Métodos para Insertar CSS en HTML

Existen tres métodos principales para insertar CSS en HTML: en línea, en el '<head>' del documento y a través de un archivo CSS externo.

B.2.1. CSS en Línea

El CSS en línea se aplica directamente a un elemento HTML utilizando el atributo `style`. Este método es útil para aplicar estilos únicos a elementos específicos, pero no es recomendado para grandes proyectos debido a la falta de reutilización y mantenimiento.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<h1 style="color: white; text-align: center;">Este es un encabezado</h1>
<p style="font-family: verdana; font-size: 20px;">Este es un parrafo.</p>

</body>
</html>
```

B.2.2. CSS en el head del Documento

El CSS también puede ser insertado dentro del elemento `<style>` en el `<head>` del documento HTML. Este método es más organizado y reutilizable que el CSS en línea.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            background-color: lightblue;
        }

        h1 {
            color: white;
            text-align: center;
        }

        p {
            font-family: verdana;
            font-size: 20px;
        }
    </style>
</head>
<body>

<h1>Este es un encabezado</h1>
<p>Este es un parrafo.</p>

</body>
</html>
```

B.2.3. CSS Externo

La forma más recomendada de aplicar CSS es utilizando un archivo CSS externo. Esto permite mantener el CSS separado del HTML, lo que facilita la lectura y el mantenimiento del código. El archivo CSS se enlaza al documento HTML mediante el elemento `<link>`.

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="styles.css">
    </head>
    <body>
```

```
        <h1>Este es un encabezado</h1>
        <p>Este es un parrafo.</p>

    </body>
</html>
```

Y el contenido del archivo `styles.css` podría ser el siguiente:

```
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
```

B.3. Conclusión

CSS es una herramienta poderosa que permite mejorar significativamente la apariencia de los documentos web. Con CSS, puedes controlar la presentación de múltiples páginas web a la vez y crear diseños atractivos y consistentes.

Apéndice C

El Estándar ECMAScript: La Base de JavaScript

C.1. ¿Qué es ECMAScript?

ECMAScript es un estándar de lenguaje de programación estandarizado por ECMA International. Es la base sobre la cual se construye JavaScript, así como otros lenguajes como JScript y ActionScript.

Imagina ECMAScript como un "libro de reglas"para lenguajes de programación. JavaScript es como un "dialecto"que sigue estas reglas, añadiendo sus propias características únicas.

C.2. Historia y Evolución

- **1995:** Netscape crea JavaScript.
- **1997:** ECMAScript 1 se establece como el primer estándar.
- **2009:** ECMAScript 5 (ES5) trae importantes mejoras.
- **2015:** ECMAScript 2015 (ES6) marca un hito con grandes cambios.
- **2016-presente:** Actualizaciones anuales (ES2016, ES2017, etc.).

C.3. ¿Por qué es importante ECMAScript?

1. **Consistencia:** Asegura que JavaScript funcione de manera similar en diferentes navegadores y entornos.
2. **Evolución:** Permite que el lenguaje se actualice y mejore regularmente.
3. **Compatibilidad:** Ayuda a los desarrolladores a entender qué características están disponibles en diferentes versiones.

C.4. Características Clave Introducidas en Versiones Principales

C.4.1. ECMAScript 5 (2009)

- Modo estricto (üse strict")
- Métodos de array (forEach, map, filter, etc.)
- Getters y setters en objetos

C.4.2. ECMAScript 2015 (ES6, 2015)

- let y const para declaración de variables
- Funciones flecha
- Clases
- Promesas para manejo asíncrono
- Módulos (import/export)
- Template literals

C.4.3. Versiones Posteriores (2016+)

- Operador de exponenciación ** (ES2016)
- Async/await para manejo asíncrono (ES2017)
- Operador de propagación para objetos (ES2018)
- Array.flat() y Array.flatMap() (ES2019)
- Operador de coalescencia nula ?? (ES2020)

C.5. Cómo ECMAScript Afecta a los Desarrolladores

1. **Aprendizaje Continuo:** Los desarrolladores deben mantenerse actualizados con las nuevas características.
2. **Compatibilidad:** Deben considerar qué versiones de ECMAScript son soportadas por sus usuarios.
3. **Herramientas:** Uso de transpiladores como Babel para usar características nuevas en entornos antiguos.
4. **Mejores Prácticas:** Cada versión suele traer formas más eficientes o legibles de escribir código.

C.6. Ejemplo Práctico: Evolución de una Función

Veamos cómo una simple función ha evolucionado a través de diferentes versiones de ECMAScript:

```
// ES3 (1999)
function sumar(a, b) {
    return a + b;
}

// ES5 (2009)
var sumar = function(a, b) {
    'use strict';
    return a + b;
};

// ES2015 (ES6)
const sumar = (a, b) => a + b;

// ES2018 con parametros por defecto
const sumar = (a = 0, b = 0) => a + b;
```

C.7. El Futuro de ECMAScript

ECMAScript continúa evolucionando. Cada año, nuevas propuestas pasan por un proceso de cuatro etapas antes de ser incluidas en el estándar. Esto permite que el lenguaje se mantenga moderno y relevante, adaptándose a las necesidades cambiantes de los desarrolladores y la web.

C.8. Conclusión

Entender ECMAScript es fundamental para cualquier desarrollador de JavaScript. No solo proporciona el contexto histórico y técnico del lenguaje, sino que también ayuda a anticipar y adaptarse a futuros cambios. Al mantenerse al día con ECMAScript, los desarrolladores pueden escribir código más eficiente, legible y moderno.

Apéndice D

Introducción a la Programación Funcional en JavaScript

La programación funcional (PF) es un paradigma que trata la computación como la evaluación de funciones matemáticas y evita el cambio de estado y datos mutables. JavaScript, aunque no es un lenguaje puramente funcional, soporta muchos conceptos de la PF.

D.1. Conceptos Clave

D.1.1. Funciones de Primera Clase

En JavaScript, las funciones son tratadas como cualquier otra variable.

```
const saludar = function(nombre) {
    return `Hola, ${nombre}!`;
};

const resultado = saludar("Maria");
console.log(resultado); // Hola, Maria!
```

D.1.2. Funciones Puras

Son funciones que, dado el mismo input, siempre producen el mismo output y no tienen efectos secundarios.

```
// Funcion pura
function sumar(a, b) {
    return a + b;
}

// Funcion impura (depende del estado externo)
let total = 0;
function sumarAlTotal(valor) {
    total += valor;
    return total;
}
```

D.1.3. Inmutabilidad

Evitar cambiar el estado de los datos después de su creación.

```
const arr1 = [1, 2, 3];
// En lugar de modificar arr1, creamos un nuevo array
const arr2 = [...arr1, 4];
console.log(arr1); // [1, 2, 3]
console.log(arr2); // [1, 2, 3, 4]
```

D.2. Técnicas de Programación Funcional

D.2.1. Map, Filter, Reduce

Estas son funciones de orden superior que operan en arrays de manera funcional.

```
const numeros = [1, 2, 3, 4, 5];

// Map: transforma cada elemento
const duplicados = numeros.map(x => x * 2);
console.log(duplicados); // [2, 4, 6, 8, 10]

// Filter: selecciona elementos basados en una condicion
const pares = numeros.filter(x => x % 2 === 0);
console.log(pares); // [2, 4]

// Reduce: combina elementos en un solo valor
const suma = numeros.reduce((acc, cur) => acc + cur, 0);
console.log(suma); // 15
```

D.2.2. Composición de Funciones

Combinar funciones simples para construir funciones más complejas.

```
const sumar5 = x => x + 5;
const multiplicarPor2 = x => x * 2;
const restar3 = x => x - 3;

const calcular = x => restar3(multiplicarPor2(sumar5(x)));
console.log(calcular(10)); // ((10 + 5) * 2) - 3 = 27
```

D.3. Ventajas de la Programación Funcional

- Código más predecible y fácil de probar
- Mejor manejo de la concurrencia
- Mayor facilidad para razonar sobre el código
- Reducción de efectos secundarios

La programación funcional en JavaScript ofrece una poderosa alternativa o complemento a la programación orientada a objetos, permitiendo escribir código más limpio, modular y mantenible.