DESARROLLO FRONTEND UTILIZANDO ECMASCRIPT

JavaScript: Fundamento Dinámico para Diversas Aplicaciones

JavaScript es un lenguaje de programación multiparadigma, dinámico y basado en prototipos. Su flexibilidad permite la implementación de estilos de programación funcional, orientada a objetos e imperativa. Originalmente concebido en 1995 para incorporar interactividad en el lado del cliente en navegadores web, su funcionalidad se ha expandido significativamente con la introducción de Node.js, lo que ha trascendido su uso más allá del entorno del navegador.

Actualmente, JavaScript es un lenguaje de programación de propósito general, aplicable en un amplio espectro de contextos:

- **Aplicaciones de Servidor (Backend):** Mediante Node.js, JavaScript es apto para ejecutar lógica de negocio y gestionar bases de datos, facilitando el desarrollo de interfaces de programación de aplicaciones (APIs) y servicios web completos.
- **Aplicaciones Móviles:** Frameworks como React Native o NativeScript posibilitan la construcción de aplicaciones nativas para sistemas operativos iOS y Android utilizando JavaScript.
- **Aplicaciones de Escritorio:** Herramientas como Electron permiten encapsular aplicaciones web (HTML, CSS, JavaScript) en aplicaciones de escritorio multiplataforma, como VS Code o Slack.
- **Aplicaciones de Consola:** Es posible crear scripts automatizados y herramientas de línea de comandos empleando JavaScript ejecutado en Node.js.
- **Robótica e Internet de las Cosas (IoT):** JavaScript está siendo implementado para el control de hardware y dispositivos conectados.
- **Aplicaciones Empotradas (Smart TV, Dispositivos Inteligentes):** Múltiples plataformas de Smart TV y dispositivos inteligentes emplean JavaScript para sus interfaces de usuario.

La prevalencia de JavaScript se evidencia en su constante posicionamiento como uno de los lenguajes de programación más populares y demandados a nivel global, congregando una extensa comunidad de profesionales.

Estándares y Motores de JavaScript: Coherencia y Ejecución

La estandarización de JavaScript es esencial para su interoperabilidad. ECMAScript es el estándar oficial que rige la especificación del lenguaje. Desde su versión inicial, ECMAScript ha experimentado una evolución constante, con lanzamientos anuales que introducen nuevas características, tales como ECMAScript 2015 (ES6), ES2016 y subsiguientes. Esta estandarización asegura la compatibilidad y predictibilidad del código JavaScript en diversos entornos.

La ejecución de JavaScript en distintos navegadores y plataformas se logra mediante los motores de JavaScript. Cada navegador desarrolla su propio motor, lo que puede generar variaciones sutiles en el rendimiento o en la implementación de ciertas funcionalidades. Algunos de los motores más relevantes son:

- SpiderMonkey: Primer motor de JavaScript, desarrollado en C por Netscape y actualmente utilizado por Mozilla Firefox.
- Rhino: Motor de JavaScript implementado en Java, comúnmente empleado para integrar JavaScript en aplicaciones Java.
- V8: Desarrollado por Google en C++, es uno de los motores más rápidos y eficientes, fundamental para Google Chrome y, de manera crítica, para Node.js.
- JavaScriptCore: Motor de JavaScript empleado en navegadores basados en WebKit, como Apple Safari.
- Carakan: Motor de JavaScript que fue utilizado por el navegador Opera antes de la adopción del motor V8.

La creación de Node.js marcó un avance crucial en la expansión de JavaScript. Node.js es un entorno de ejecución que posibilita la ejecución de JavaScript fuera del navegador, específicamente en el servidor. Su característica distintiva es estar construido sobre el motor V8 de Chrome, lo que le confiere una velocidad y eficiencia notables para operaciones de entrada/salida no bloqueantes, estableciendo a JavaScript como una opción viable y potente para el desarrollo backend.

Sintaxis y Variables: Fundamentos de la Codificación en JavaScript

La sintaxis de JavaScript exhibe influencias de lenguajes como Java, Awk, Perl y Python, lo que la hace familiar para muchos desarrolladores.

Aspectos Sintácticos Clave:

- **Sentencias:** Las instrucciones en JavaScript se denominan sentencias y se separan convencionalmente por un punto y coma (;). Aunque en múltiples ocasiones el punto y coma es opcional (cuando la sentencia ocupa una línea individual), su uso explícito se considera una buena práctica que mejora la legibilidad y previene errores en determinados escenarios.
- **Sensibilidad a Mayúsculas y Minúsculas:** JavaScript es un lenguaje case-sensitive, lo que implica que miVariable y mivariable son identificadores distintos.
- **Conjunto de Caracteres:** Soporta el conjunto de caracteres Unicode, permitiendo la utilización de una vasta gama de símbolos y caracteres internacionales.

Formas de Integrar JavaScript en un Documento HTML:

Existen tres métodos principales para incorporar código JavaScript en una página web:

- **Integración Directa (Inline):** El código JavaScript se incrusta directamente en el documento HTML dentro de las etiquetas <script>. Este método es adecuado para scripts concisos y específicos de una sección de la página.

HTML

<script>

console.log("Hola desde el script inline");

</script>

- Archivos Externos: Es la práctica más recomendada para proyectos de mayor envergadura y con una estructura organizada. El código JavaScript se guarda en un archivo separado (ej., app.js) y se vincula al documento HTML mediante el atributo src de la etiqueta <script>. Esto optimiza la modularidad, el almacenamiento en caché del navegador y la mantenibilidad.

HTML

<script src="path/to/tu_script.js"></script>

- Manejadores de Eventos en Línea (Inline Event Handlers): El código JavaScript se asocia directamente a un atributo de evento en una etiqueta HTML (ej., onclick, onmouseover). Esta práctica es generalmente considerada menos óptima y más difícil de mantener que las anteriores, aunque funcional para interacciones muy sencillas.

HTML

<button onclick="alert('Botón presionado!')">Haz clic</button>

Variables en JavaScript: Almacenamiento de Datos

Las variables en JavaScript son identificadores que almacenan valores en la memoria. El nombre de una variable debe comenzar con una letra, un guión bajo (_) o un símbolo de dólar ($). La declaración de variables en JavaScript ha evolucionado:

- **var:** La forma tradicional de declarar variables, con un ámbito de función (o global si se declara fuera de cualquier función). Presenta particularidades como el hoisting que pueden generar comportamientos inesperados si no se comprenden.
- **let (Introducido en ES6):** Declara variables con un ámbito de bloque, lo que implica que solo son accesibles dentro del bloque de código donde

fueron definidas (ej., dentro de una sentencia if o un bucle for). Es la forma preferida para variables cuyo valor puede cambiar.

- **const (Introducido en ES6):** Declara una constante, lo que significa que su valor no puede ser reasignado una vez inicializado. También posee un ámbito de bloque. Es ideal para valores que no se modificarán durante la ejecución.

Tipos de Ámbito de Variables:

- **Variables Globales:** Declaradas sin un ámbito específico (o con var fuera de una función), se convierten en propiedades del objeto global (window en navegadores, global en Node.js) y son accesibles desde cualquier parte del código.
- **Variables Locales:** Declaradas dentro de una función o un bloque de código (let, const). Su alcance está restringido a ese contexto y solo son accesibles dentro del mismo.

Tipos de Datos en JavaScript:

JavaScript es un lenguaje de tipado dinámico, lo que significa que el tipo de una variable se determina en tiempo de ejecución. Define ocho tipos de datos principales:

- **Siete Tipos Primitivos:**
    - **Boolean:** Representa un valor lógico (true o false).
    - **Null:** Representa la ausencia intencionada de cualquier valor de objeto. Es un valor singular sin un tipo asociado.
    - **Undefined:** Representa una variable que ha sido declarada pero a la que aún no se le ha asignado un valor.
    - **Number:** Representa valores numéricos enteros o de punto flotante.
    - **BigInt:** Permite representar números enteros con una precisión arbitrariamente grande, superando el límite de Number.
    - **String:** Representa secuencias de caracteres (texto).
    - **Symbol (Introducido en ES6):** Representa un identificador único e inmutable, frecuentemente utilizado para claves de propiedades de objeto que no se espera que generen colisiones.
- **Un Tipo Complejo:**
    - **Object:** El tipo fundamental para estructuras de datos más complejas, como arrays, funciones y objetos literales.

Estructuras de Control y Bucles: Regulación del Flujo del Programa

JavaScript proporciona estructuras de control esenciales para determinar la lógica y el flujo de ejecución de un programa.

Estructuras Condicionales:

- **if...else if...else:** Permite ejecutar diferentes bloques de código basándose en la evaluación de una o varias condiciones booleanas.

JavaScript

```
if (condicion1) {
   // Código si se cumple condicion1
} else if (condicion2) {
   // Código si se cumple condicion2
} else {
   // Código si no se cumple ninguna de las anteriores
}
```

- **switch:** Una estructura de control que permite ejecutar diferentes bloques de código en función del valor de una expresión, ofreciendo una alternativa más clara a múltiples sentencias if/else if anidadas cuando se evalúan muchas condiciones posibles basadas en un único valor.

JavaScript

```
switch (expresion) {
   case valor1:
      // Código para valor1
      break;
   case valor2:
      // Código para valor2
      break;
   default:
      // Código si ningún caso coincide
}
```

Estructuras de Bucle (Iteración):

Permiten la repetición de la ejecución de un bloque de código un número determinado de veces o mientras se cumpla una condición.

- **for:** Ideal para iterar un número conocido de veces.

JavaScript

```
for (let i = 0; i < 10; i++) {
    // Código que se repite 10 veces
}
```

- **while:** Repite un bloque de código mientras una condición sea verdadera. La condición se evalúa antes de cada iteración.

JavaScript

```
while (condicion) {
    // Código que se repite
}
```

- **do...while:** Similar a while, pero asegura que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa después de la primera iteración.

JavaScript

```
do {
    // Código que se repite
} while (condicion);
```

- **forEach (Método de Array):** Un método de orden superior para arrays que ejecuta una función proporcionada una vez por cada elemento del array. No devuelve un nuevo array y no interrumpe el bucle.

JavaScript

```
array.forEach(elemento => {
    // Código para cada elemento
});
```

- **for...in:** Itera sobre las propiedades enumerables de un objeto (claves). Utilizado principalmente para objetos, no se recomienda para arrays debido a que puede iterar sobre propiedades inesperadas o en un orden no garantizado.

JavaScript

```
for (const clave in objeto) {
    // Código para cada clave
}
```

- **for...of (Introducido en ES6):** Itera sobre los valores de objetos iterables (como arrays, strings, Map, Set). Es la forma preferida para recorrer los elementos de colecciones.

JavaScript

```
for (const valor of iterable) {
   // Código para cada valor
}
```

Funciones en JavaScript: Módulos de Lógica Reutilizable

Las funciones son bloques de código reutilizables y autónomos diseñados para ejecutar una tarea específica. Son un pilar fundamental en JavaScript para organizar el código, promover la reutilización y mejorar la legibilidad.

Definición y Estructura:

Una función se define con la palabra clave function, seguida de un nombre (opcional en ciertos casos), una lista de parámetros entre paréntesis (los datos que la función puede recibir) y un bloque de código entre llaves {} que contiene las instrucciones a ejecutar.

JavaScript

```
function nombreDeMiFuncion(parametro1, parametro2) {
   // Código a ejecutar
   return resultado; // Opcional: devuelve un valor
}
```

Las funciones pueden retornar un valor utilizando la palabra clave return. Si una función no posee una sentencia return explícita, implícitamente retorna undefined.

Formas de Invocar (Llamar) una Función:

- **Desde el Código JavaScript:** La forma más común es simplemente llamar a la función por su nombre, pasando los argumentos necesarios.

JavaScript

```
let miResultado = nombreDeMiFuncion(valor1, valor2);
```

- **Desde un Evento HTML:** Asociando la función a un evento específico de un elemento HTML (aunque esta práctica es menos recomendable para aplicaciones complejas).

HTML

```
<button onclick="nombreDeMiFuncion('Hola')">Haz clic</button>
```

- **Funciones Auto-invocadas (Immediately Invoked Function Expressions - IIFE):** Son funciones que se definen y se ejecutan inmediatamente después de su creación. Son útiles para establecer un ámbito privado y evitar la contaminación del ámbito global.

JavaScript

```
(function () {
   // Código que se ejecuta inmediatamente
   console.log("Esta función se auto-invocó.");
})();
```

Tipos de Funciones Avanzadas (ES6+):

- **Funciones Anónimas:** Funciones que carecen de un nombre. Frecuentemente se utilizan como callbacks o se asignan a variables.

JavaScript

```
const miFuncionAnonima = function() {
   console.log("Soy una función anónima.");
};
```

- **Funciones Flecha (Arrow Functions - Introducidas en ES6):** Una sintaxis más concisa y moderna para definir funciones anónimas. Son particularmente útiles para callbacks y presentan un comportamiento distinto respecto al contexto de this.

JavaScript

```
const miFuncionFlecha = (parametro) => {
   console.log(`Valor: ${parametro}`);
};
```

- **Constructor Function() (Desaconsejado para uso general):** Permite crear funciones a partir de cadenas de texto. Generalmente no se recomienda por razones de seguridad (inyección de código) y rendimiento.

Manejo de Cadenas (Strings) y Arrays: Colecciones Esenciales

JavaScript ofrece capacidades robustas para la manipulación de texto y colecciones de datos.

Cadenas de Texto (Strings):

Las **cadenas de texto** son secuencias de caracteres y pueden ser creadas utilizando comillas simples ('...'), comillas dobles ("...") o backticks (`...`).

- **Objeto String:** JavaScript proporciona un objeto String incorporado que ofrece numerosos métodos para manipular cadenas (ej., charAt(), indexOf(), substring(), split(), toUpperCase(), toLowerCase()).
- **Plantillas Literales (Template Literals - Introducidas en ES6):** Permiten incrustar expresiones y variables directamente en las cadenas de texto utilizando el carácter backtick (`). También facilitan la creación de cadenas de varias líneas.

JavaScript

```
const nombre = "Juan";

const saludo = `Hola, mi nombre es ${nombre}.`; // "Hola, mi nombre es Juan."
```

Arrays (Arreglos):

Los arrays son colecciones ordenadas de datos que pueden contener valores de cualquier tipo, incluyendo otros arrays u objetos. Son fundamentales para almacenar listas de elementos.

- **Creación:** Se pueden crear utilizando corchetes [] (notación literal) o el constructor Array.

JavaScript

```
const miArray = [1, 'hola', true, [4, 5]];
```

- **Objeto Array:** El objeto Array incorporado ofrece una rica colección de métodos para manipular arrays (ej., sort(), pop(), push(), shift(), unshift(), splice(), slice(), concat(), join(), map(), filter(), reduce(), find(), findIndex()).

Objetos en JavaScript: Colecciones de Propiedades Flexibles

Los objetos en JavaScript son colecciones de pares clave-valor, lo que los hace análogos a los diccionarios en Python, los mapas en Java o los hashes en Ruby. Son esenciales para representar estructuras de datos complejas.

- **Creación:** Se pueden crear utilizando llaves {} (notación literal, la más común) o el constructor Object.

JavaScript

```
const persona = {
    nombre: "Ana",
    edad: 30,
    esEstudiante: false
};
```

- **Acceso a Propiedades:** Se puede acceder a los valores de las propiedades de un objeto utilizando:
    - **Notación de Punto:** objeto.propiedad (ej., persona.nombre). Preferida cuando el nombre de la propiedad es conocido y válido como identificador.
    - **Notación de Corchetes/Array:** objeto['propiedad'] (ej., persona['edad']). Útil cuando el nombre de la propiedad es dinámico (una variable) o contiene caracteres especiales.
- **Desestructuración de Objetos (Introducida en ES6):** Permite "desempacar" valores de las propiedades de un objeto directamente en variables independientes, optimizando la concisión y legibilidad del código.

JavaScript

```
const { nombre, edad } = persona; // nombre = "Ana", edad = 30
```

Clases en JavaScript: Sintaxis Simplificada para Programación Orientada a Objetos

A partir de ECMAScript 6 (ES6), JavaScript introdujo la palabra clave class como una sintaxis de azúcar para la programación orientada a objetos basada en prototipos preexistente. Aunque internamente las clases en JavaScript operan bajo el modelo de prototipos, la sintaxis class hace que la herencia y la definición de objetos se asemejen más a los modelos tradicionales de programación orientada a objetos de otros lenguajes.

- **Definición de Clases:** Una clase permite definir un constructor (un método especial que se ejecuta cuando se crea una nueva instancia de la clase) y otros métodos que encapsulan el comportamiento.

JavaScript

```
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
  }

  saludar() {
    console.log(`Hola, soy ${this.nombre}.`);
  }
```

}

- **Herencia:** La herencia entre clases se implementa utilizando la palabra clave extends para indicar que una clase hereda de otra. El método super() se emplea dentro del constructor de la clase hija para invocar al constructor de la clase padre, asegurando la correcta inicialización de las propiedades de la clase base.

JavaScript

```
class Estudiante extends Persona {
    constructor(nombre, edad, carrera) {
        super(nombre, edad); // Llama al constructor de Persona
        this.carrera = carrera;
    }

    presentarse() {
        console.log(`Soy ${this.nombre}, estudio ${this.carrera}.`);
    }
}
```

Colecciones de Datos Avanzadas: Map y Set (Introducidos en ES6)

ES6 enriqueció el lenguaje con dos estructuras de datos novedosas y altamente útiles para gestionar colecciones de manera más eficiente y versátil:

- **Map:** Es una colección de pares clave-valor, similar a los objetos, pero con dos ventajas distintivas:
    - Permite que las claves sean de cualquier tipo de dato (números, objetos, funciones, etc.), no solo cadenas o símbolos como en los objetos literales.
    - Mantiene el orden de inserción de los elementos. Ofrece métodos dedicados para operaciones comunes: set(), get(), has(), delete(), size.

JavaScript

```
const miMapa = new Map();
miMapa.set('nombre', 'Carlos');
miMapa.set(1, 'Uno');
console.log(miMapa.get('nombre')); // Carlos
```

- **Set:** Es una colección de valores únicos, lo que significa que cada elemento solo puede aparecer una vez en el Set. No mantiene un orden específico y no tiene claves. Es ideal para almacenar listas de elementos sin duplicados. Ofrece métodos como add(), delete(), has(), size.

JavaScript

```
const miSet = new Set();

miSet.add(1);

miSet.add(5);

miSet.add(1); // No se añade, ya existe

console.log(miSet.has(5)); // true
```

Canvas para Gráficos: Dibujo en el Navegador

El elemento <canvas> en HTML5 proporciona una API de JavaScript para dibujar gráficos 2D y 3D directamente en el navegador. Es un área de dibujo en el documento HTML que se controla íntegramente con JavaScript.

- Funcionamiento: Se obtiene un contexto de dibujo del elemento <canvas> (generalmente un contexto 2D) y luego se utilizan los métodos de este contexto para dibujar formas, texto, imágenes y aplicar transformaciones.
- Coordenadas: El canvas se representa como una cuadrícula bidimensional. El punto (0,0) se ubica en la esquina superior izquierda del canvas.
- Métodos Comunes de Dibujo:
    - moveTo(x, y): Mueve el "lápiz" a las coordenadas (x, y) sin dibujar.
    - lineTo(x, y): Dibuja una línea desde la posición actual del lápiz hasta (x, y).
    - arc(x, y, radio, anguloInicio, anguloFin, sentido): Dibuja un arco o un círculo.
    - rect(x, y, ancho, alto): Dibuja un rectángulo.
    - stroke(): Dibuja el contorno del trazo actual.
    - fill(): Rellena el trazo actual.
    - fillText(texto, x, y): Dibuja texto relleno.
    - strokeText(texto, x, y): Dibuja el contorno del texto.
    - beginPath(), closePath(): Para iniciar y finalizar rutas de dibujo.

El <canvas> es una herramienta potente para visualizaciones de datos, juegos web, editores de imágenes en el navegador y cualquier aplicación que requiera un control preciso del renderizado gráfico a nivel de píxel.

Glosario de Términos Clave

- ECMAScript: La especificación estándar oficial del lenguaje JavaScript, en constante evolución con nuevas versiones anuales.
- Motor de JavaScript: Un programa o intérprete que ejecuta código JavaScript. Cada navegador o entorno (Node.js) posee su propio motor, lo que puede influir en la compatibilidad y el rendimiento.
- Node.js: Un entorno de ejecución de JavaScript de código abierto, construido sobre el motor V8 de Chrome, que posibilita la ejecución de código JavaScript del lado del servidor.
- Variable: Un contenedor identificado para almacenar valores en la memoria, cuyo tipo puede variar dinámicamente. Puede ser global, local o constante (const).
- Tipos de Datos: Las categorías de valores que JavaScript puede manipular, incluyendo primitivos (Boolean, Null, Undefined, Number, BigInt, String, Symbol) y el tipo complejo Object.
- Función: Un bloque de código reutilizable diseñado para llevar a cabo una tarea específica, con la capacidad de aceptar parámetros y devolver valores.
- Cadena (String): Una secuencia de caracteres que representa texto, inmutable y con métodos para su manipulación.
- Array (Arreglo): Una colección ordenada de valores, que puede contener elementos de cualquier tipo.
- Objeto: Una colección de pares clave-valor que permite representar estructuras de datos complejas y relacionadas.
- Clase: Una sintaxis para crear "plantillas" de objetos en JavaScript, definiendo constructores y métodos, aunque internamente se fundamenta en el modelo de prototipos.
- Map: Una colección de pares clave-valor donde las claves pueden ser de cualquier tipo de dato, manteniendo el orden de inserción.
- Set: Una colección de valores únicos, sin claves ni orden garantizado.
- Canvas: Un elemento HTML5 que proporciona una API para dibujar gráficos rasterizados (basados en píxeles) directamente en el navegador utilizando JavaScript.

Procedimientos Fundamentales

- Inclusión de JavaScript en HTML: Integrar código JavaScript en un documento HTML mediante la etiqueta <script> (directamente o con un archivo externo) o a través de atributos de evento en línea.
- Declaración de Variables: Emplear las palabras clave var, let o const para crear variables con el ámbito apropiado (global, función o bloque) y asignarles valores.
- Definición de Funciones: Estructurar bloques de código reutilizables utilizando la palabra clave function (o las sintaxis de función flecha), especificando parámetros y el retorno de valores.
- Invocación de Funciones: Ejecutar una función llamándola por su nombre y pasando los argumentos necesarios, ya sea desde el propio código JavaScript o mediante eventos HTML.
- Manipulación de Cadenas: Utilizar los métodos del objeto String (ej., charAt(), indexOf(), substring(), split(), trim()) y las plantillas literales para procesar y formatear texto.
- Manipulación de Arrays: Emplear los métodos del objeto Array (ej., push(), pop(), map(), filter(), reduce(), sort(), splice()) para agregar, eliminar, transformar y ordenar elementos en colecciones.
- Acceso a Propiedades de Objetos: Obtener o modificar valores de propiedades de objetos utilizando la notación de punto (.) o la notación de corchetes ([]).
- Creación de Clases: Definir estructuras de objetos mediante la palabra clave class, incluyendo un constructor y métodos para encapsular comportamiento y datos.
- Dibujo en Canvas: Obtener el contexto de renderizado de un elemento <canvas> y utilizar sus métodos (ej., moveTo(), lineTo(), arc(), rect(), fillText(), stroke(), fill()) para dibujar gráficos 2D.