Clase: Introducción a JavaScript y Estructuras Básicas

Duración total: 2 horas y 30 minutos

Objetivo general:

Al finalizar la clase, los estudiantes comprenderán los fundamentos de JavaScript, incluyendo la sintaxis básica, tipos de datos, estructuras de control y la creación de funciones. Se hará un análisis profundo de cómo funcionan los bloques de código y el entorno de ejecución en el navegador.

---

1. Introducción a JavaScript (30 minutos)

¿Qué es JavaScript? JavaScript es un lenguaje de programación que sigue el paradigma **orientado a objetos** basado en prototipos, y es **interpretado**, lo que significa que el código se ejecuta directamente por el motor JavaScript del navegador sin necesidad de un paso de compilación. Este lenguaje es un pilar en el desarrollo web junto con **HTML** y **CSS**.

¿Por qué es interpretado? Cuando el navegador recibe el código JavaScript junto con el HTML y CSS, el motor JavaScript (como V8 en Chrome) lo ejecuta directamente, línea por línea, sin necesidad de generar un archivo intermedio compilado. Esto permite actualizaciones dinámicas e interactivas del contenido web sin necesidad de recargar la página.

Aplicaciones de JavaScript: JavaScript se usa principalmente en el desarrollo frontend, pero con la aparición de Node.js, también ha ganado presencia en el backend. En el contexto de esta clase, nos enfocamos en el lado del cliente.

Motores JavaScript:

- **V8** en Chrome.
- **SpiderMonkey** en Firefox.
- **Chakra** en Edge.

Actividad práctica:

1. Abrir la consola del navegador (F12 en la mayoría de navegadores).
2. Ejecutar este simple comando:

```
console.log("¡Hola, mundo!");
```

1. Esto permite ver la interacción entre el código y el navegador directamente.

---

2. Variables y Tipos de Datos en JavaScript (35 minutos)

Declaración de Variables JavaScript permite declarar variables de tres formas: var, let, y const. Entender las diferencias entre ellas es crucial para evitar errores y comprender mejor cómo las variables se comportan en diferentes contextos.

- **var**: Se utiliza desde los inicios de JavaScript, pero tiene un comportamiento peculiar: **es global o de función**, y además permite la redeclaración de la misma variable.
    - **Problema con var**: Puede generar confusión porque una variable declarada con var se "hoistea", es decir, su declaración se eleva al principio de la función o del bloque de código, pero su valor no.
- **let**: Introducido en ES6, resuelve los problemas de alcance de var. Tiene **alcance de bloque**, lo que significa que su valor solo es accesible dentro del bloque de código donde fue declarado, lo que evita errores de redeclaración o colisión de variables.
- **const**: También introducido en ES6, se usa para declarar **constantes**. Esto significa que el valor de la variable no puede cambiar después de haber sido asignado. Aunque const también tiene un **alcance de bloque**, se diferencia de let en que prohíbe la reasignación.

Ejemplos:

```
let nombre = "Juan";  // let permite reasignación
const PI = 3.1416;    // const no permite reasignación


var edad = 30;  // var es hoisted
```

Hoisting: JavaScript "eleva" las declaraciones de variables y funciones al principio de su contexto (función o bloque) antes de la ejecución del código. Sin embargo, sólo las declaraciones de variables se elevan, no sus inicializaciones.

- **Ejemplo de hoisting con var**:

```
console.log(edad);  // undefined, porque la declaración se "eleva"
var edad = 30;
```

- **let y const** no sufren de este problema de manera similar a var, pues están sujetas a un "Temporal Dead Zone" (Zona Temporal Muerta), lo que significa que no se pueden usar antes de su declaración en el código.

Tipos de Datos en JavaScript

JavaScript tiene tipos de datos **primitivos** y **complejos**:

- **Primitivos**:
  - **string**: Cadenas de texto.
  - **number**: Números (tanto enteros como decimales).
  - **boolean**: Valores lógicos (true o false).
  - **null**: Intencionalmente sin valor.
  - **undefined**: Variable que ha sido declarada pero no inicializada.
- **Tipos complejos**:
  - **Objetos**: Contienen pares clave-valor.
  - **Arrays**: Listas ordenadas de valores.

Ejemplos:

```
let nombre = "Ana";  // string
const edad = 30;     // number
let esMayor = true;  // boolean
let direccion = null; // null
```

```
let telefono;        // undefined
```

Actividad práctica:

1. Los estudiantes deben declarar variables utilizando let, const, y var, y verificar su comportamiento en la consola, viendo cómo afecta el alcance (scope) de la variable.

---

3. Estructuras de Control y Bucles en JavaScript (45 minutos)

Condicionales (if, else, switch)

Las estructuras condicionales permiten que el programa tome decisiones basadas en expresiones lógicas.

- **Condicional if-else**: Es una de las estructuras más básicas. Evalúa una condición, y si esta es verdadera, ejecuta un bloque de código. Si es falsa, puede ejecutar otro bloque de código usando else.
    - **Ejemplo**:

```
let edad = 20;
if (edad >= 18) {
  console.log("Es mayor de edad");
} else {
  console.log("Es menor de edad");
}
```

switch: Una alternativa a las estructuras if-else múltiples, usada para ejecutar código basado en el valor de una variable.

- Ejemplo de switch:

```
let dia = 3;
switch (dia) {
```

```
  case 1:
    console.log("Lunes");
    break;
  case 2:
    console.log("Martes");
    break;
  case 3:
    console.log("Miércoles");
    break;
  default:
    console.log("Día no válido");
}
```

Bucles (for, while)

Los bucles son esenciales para ejecutar repetidamente un bloque de código.

- **Bucle for**: Ideal cuando sabes cuántas veces necesitas ejecutar el bloque de código.

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

Bucle while: Ejecuta el bloque de código mientras la condición sea verdadera.

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

```
}
```

Actividad práctica:

1. Crear un bucle for que cuente de 1 a 10 y muestre los números en la consola.
2. Escribir un switch que devuelva el nombre del día de la semana según un número.

---

4. Funciones en JavaScript (40 minutos)

Declaración de Funciones

Una función es un bloque de código que realiza una tarea específica y puede ser reutilizado. Hay dos maneras comunes de declararlas:

- **Función tradicional**:

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

Funciones de Flecha (ES6): Las funciones de flecha introducidas en ES6 ofrecen una sintaxis más corta y manejan el contexto de this de manera diferente.

```
const sumar = (a, b) => a + b;
```

Alcance o Scope

El alcance determina la accesibilidad de las variables. En JavaScript, hay dos tipos principales de alcance:

- **Global**: Las variables declaradas fuera de una función están disponibles en todo el código.
- **Local**: Las variables declaradas dentro de una función solo están disponibles dentro de esa función.

Ejemplo:

```
let nombre = "Pedro";  // Global

function saludar() {
    let saludo = "Hola";  // Local
    console.log(`${saludo}, ${nombre}`);
}

saludar();  // Hola, Pedro
console.log(saludo);  // Error: 'saludo' no está definido
```

Actividad práctica:

1. Crear una función que acepte dos parámetros y devuelva su suma.
2. Reescribir esa función como una función de flecha.

Recapitulación y Preguntas (20 minutos)

Para cerrar la clase, se hará una recapitulación de todos los conceptos aprendidos durante la sesión. Este es un momento clave para reforzar los puntos más importantes, permitiendo que los estudiantes consoliden sus conocimientos y hagan preguntas sobre los conceptos que no hayan quedado claros.

Recapitulación de conceptos clave:

1. **Variables en JavaScript**:
    - Diferencias entre var, let y const.
    - El concepto de **hoisting** y cómo afecta a var.
    - Alcance de las variables: **global** vs **local**.
2. **Tipos de datos**:

- Primitivos (string, number, boolean, null, undefined).
- Diferencia entre **primitivos** y **complejos** (objetos y arrays).

3. **Estructuras de control**:
    - Condicionales (if, else, switch) para tomar decisiones.
    - Bucles (for, while) para ejecutar código repetidamente.
4. **Funciones**:
    - Declaración de funciones tradicionales.
    - Uso de funciones de flecha (ES6).
    - El concepto de **scope** (alcance) y cómo afecta a las variables dentro de funciones.

Actividad final (10 minutos)

Para poner en práctica todo lo aprendido, los estudiantes realizarán un ejercicio que combine los temas principales vistos en clase:

Mini proyecto:

1. **Objetivo**: Crear un pequeño programa que use variables, estructuras de control, bucles y funciones.
    - **Paso 1**: Declarar una variable que almacene la edad de una persona.
    - **Paso 2**: Usar un condicional (if-else) para verificar si la persona es mayor de edad.
    - **Paso 3**: Usar un bucle for para contar desde 1 hasta la edad de la persona.
    - **Paso 4**: Crear una función que tome un número como parámetro y devuelva su cuadrado. Aplicar esta función sobre la edad y mostrar el resultado.

```
let edad = 25;
```

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

for (let i = 1; i <= edad; i++) {
    console.log(`Contando: ${i}`);
}

const cuadrado = (numero) => numero * numero;
console.log(`El cuadrado de la edad es: ${cuadrado(edad)}`);
```

Explicación del código:

- Se utiliza let para declarar la variable edad.
- Se usa un condicional if-else para verificar si es mayor de edad.
- Se aplica un bucle for para contar hasta la edad.
- Se implementa una función de flecha para calcular el cuadrado de la edad.

Material adicional y ejercicios recomendados para casa

Se recomendarán recursos y ejercicios para que los estudiantes sigan practicando en su tiempo libre:

- **Plataformas recomendadas**:
    - Codewars: Para practicar problemas de programación en diferentes niveles de dificultad.

    - LeetCode: Para ejercicios más avanzados, especialmente si desean prepararse para entrevistas técnicas.

- **Documentación oficial**:
    - MDN Web Docs: JavaScript: Referencia técnica y guía completa sobre JavaScript.