**JavaScript Basics**

**Objective:** Students will learn the basics of JavaScript, including variables, data types, and basic operations.

**Introduction to JavaScript**

**What is JavaScript?**

**Brief history of JavaScript**

JavaScript was created in 1995 by Brendan Eich while working at Netscape Communications Corporation. Initially developed in just 10 days, JavaScript was designed to make web pages more interactive. Despite its rushed creation, JavaScript has evolved significantly and is now one of the core technologies of the World Wide Web, alongside HTML and CSS.

**Importance of JavaScript in web development**

JavaScript plays a crucial role in web development. While HTML structures content and CSS styles it, JavaScript adds interactivity and dynamic behaviour. It allows developers to create rich, interactive web applications and enhances user experience by enabling features such as form validation, dynamic content updates, animations, and more.

**How JavaScript differs from HTML and CSS**

HTML (HyperText Markup Language) provides the structure and content of a web page. CSS (Cascading Style Sheets) defines the presentation and layout of the content. JavaScript, on the other hand, is a programming language that enables dynamic interactions. It can manipulate HTML and CSS to create interactive effects, respond to user actions, and communicate with servers.

**The Role of JavaScript in Web Development**

Enhancing interactivity and user experience

**Examples** of JavaScript usage on popular websites (e.g., dynamic content, form validation, animations)

**Embedding JavaScript in HTML** (Internal vs. External Scripts)

**Internal JavaScript:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Internal JavaScript</title>

<script>

console.log('Hello from internal JavaScript!');

</script>

</head>

<body>

<h1>Internal JavaScript Example</h1>

</body>

</html>

**External JavaScript:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>External JavaScript</title>

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

</head>

<body>

<h1>External JavaScript Example</h1>

</body>

</html>

*External javascript file uses a .js extension.*

**script.js**

console.log('Hello from external JavaScript!');

**Variables and Data Types**

**Declaring Variables using var, let, and const**

1. **var**

Declaration and Scope:

* Declares a variable.
* Function-scoped: If declared inside a function, it's only accessible within that function. If declared outside, it's globally scoped.
* Can be redeclared and updated within its scope.

Hoisting:

* Variables declared with var are hoisted to the top of their scope and initialized with undefined. This means you can reference the variable before its declaration without causing an error, although it will be undefined until the declaration is encountered.

console.log(a); // undefined

var a = 10;

console.log(a); // 10

1. **let**

Declaration and Scope:

* Declares a block-scoped variable, meaning it's only accessible within the nearest enclosing block (e.g., {}, for loops).
* Cannot be redeclared within the same scope, but can be updated.

Hoisting:

* Variables declared with let are hoisted to the top of their block but are not initialized. This creates a "temporal dead zone" from the start of the block until the declaration is encountered, making it an error to reference the variable before it's declared.

console.log(b); // ReferenceError: b is not defined

let b = 20;

console.log(b); // 20

1. **const**

Declaration and Scope:

* Declares a block-scoped constant, which must be initialized at the time of declaration.
* Cannot be redeclared or updated. However, if the constant is an object or array, the properties or elements of the object or array can be changed.

Hoisting:

* Variables declared with const are hoisted to the top of their block but are not initialized, similar to let. This also creates a "temporal dead zone."

const c = 30;

console.log(c); // 30

c = 40; // TypeError: Assignment to constant variable.

const obj = { key: 'value' };

obj.key = 'newValue'; // This is allowed

console.log(obj.key); // newValue

**var**: Function-scoped, can be redeclared and updated, hoisted and initialized with undefined.

**let**: Block-scoped, cannot be redeclared but can be updated, hoisted but not initialized, creating a temporal dead zone.

**const**: Block-scoped, cannot be redeclared or updated (but object properties can be modified), hoisted but not initialized, creating a temporal dead zone.

**Primitive Data Types:**

1. **Numbers:** Represent numerical values. JavaScript uses a single type for both integers and floating-point numbers.

let age = 25;

Var joshHeight = 30.5;

1. **Strings:** Represent textual data, enclosed in single quotes, double quotes, or backticks.

let name = 'Alice';

var fullName = “Caleb Abiola”;

1. **Booleans:** Represent logical values true or false.

let isStudent = true;

var isLegal = false;

1. **Null:** Represents the intentional absence of any object value.

let emptyValue = null;

const myValue;

1. **Undefined:** Indicates a variable has been declared but not yet assigned a value.

let undefinedValue;

**Complex Data Types:**

1. **Objects:** Collections of key-value pairs. Keys are strings (or symbols), and values can be any type.

let person = {

name: 'Alice',

age: 25

};

1. **Arrays:**  Ordered collections of values, indexed by numbers.

let fruits = ['apple', 'banana', 'cherry'];

**Basic Operations**

**Arithmetic Operations:**

Arithmetic operations allow you to perform calculations with numerical values. Common operations include addition, subtraction, multiplication, and division.

let x = 10;

let y = 5;

console.log(x + y); // 15

console.log(x - y); // 5

console.log(x \* y); // 50

console.log(x / y); // 2

**String Concatenation**

String concatenation combines multiple strings into one. This can be done using the + operator or template literals (backticks) for more complex combinations.

let firstName = 'John';

let lastName = 'Doe';

let fullName = firstName + ' ' + lastName;

console.log(fullName); // John Doe

// Using template literals let fullNameTemplate = `${firstName} ${lastName}`; console.log(fullNameTemplate); // John Doe

**Assignment Operators**

Assignment operators assign values to variables. The basic assignment operator is =, but there are also compound assignment operators for performing arithmetic operations and assignment in one step.

let a = 10;

a += 5; // equivalent to a = a + 5

console.log(a); // 15

**Comparison Operators**

Comparison operators compare two values and return a boolean (true or false). These include equality (==), strict equality (===), inequality (!=), strict inequality (!==), greater than (>), and less than (<).

== (Equal To)

=== (Strict Equal To)

!= (Not Equal To)

!== (Strict Not Equal To)

> (Greater Than)

>= (Greater Than or Equal To)

< (Less Than)

<= (Less Than or Equal To)

let num1 = 10;

let num2 = 20;

console.log(num1 == num2); // false

console.log(num1 != num2); // true

console.log(num1 > num2); // false

console.log(num1 < num2); // true

**Logical Operators**

Logical operators in JavaScript are used to perform logical operations on values, returning a Boolean value (true or false). These operators are commonly used in control flow statements and conditional expressions. Here’s a detailed overview of the logical operators available in JavaScript:

1. **Logical AND (&&)**: The logical AND operator returns true if both operands are true, otherwise it returns false. It also short-circuits, meaning if the first operand is false, it does not evaluate the second operand.

Example

let a = 5;

let b = 10;

console.log(a > 0 && b > 0); // true (both conditions are true)

console.log(a > 0 && b < 0); // false (second condition is false)

1. **Logical OR (||)**: The logical OR operator returns true if at least one of the operands is true. It also short-circuits, meaning if the first operand is true, it does not evaluate the second operand.

Example

let c = -5; let d = 10;

console.log(c > 0 || d > 0); // true (second condition is true)

console.log(c > 0 || d < 0); // false (both conditions are false)

let isTrue = true;

let isFalse = false;

console.log(isTrue && isFalse); // false

console.log(isTrue || isFalse); // true

console.log(!isTrue); // false

**Practical Exercise: Basic Calculator**

**Objective:**

Create a simple calculator using JavaScript to perform basic arithmetic operations.

**HTML Structure:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Basic Calculator</title>

</head>

<body>

<h1>Calculator</h1>

<input type="number" id="num1" placeholder="Enter first number">

<input type="number" id="num2" placeholder="Enter second number">

<button onclick="calculate()">Calculate</button>

<p id="result"></p>

<script>

function calculate() {

let num1 = parseFloat(document.getElementById('num1').value);

let num2 = parseFloat(document.getElementById('num2').value);

let sum = num1 + num2;

document.getElementById('result').innerText = 'Result: ' + sum;

}

</script>

</body>

</html>

**Explanation:**

The calculate function retrieves the values from the input fields, parses them as floats, performs the addition, and displays the result in the <p> element.