JavaScript

Unit-1

SCS183: Web Technologies-II

CO1:JavaScript Fundamentals and Integration

Definition

- JavaScript is a lightweight, interpreted, and high-level programming language primarily used for enhancing the interactivity and functionality of websites.
- It was initially designed to enable dynamic behavior on web pages, such as user interactions, animations, and content updates without requiring the page to reload.
- It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser.

History of JavaScript (Key Highlights)

- **Creation (1995)**:
  JavaScript was created by Brendan Eich at Netscape in just 10 days. Initially called **Mocha** and later **LiveScript**, it was renamed JavaScript to capitalize on Java's popularity.
- **Adoption by Microsoft (1996)**:
  Microsoft developed its own JavaScript implementation, **JScript**, leading to browser compatibility challenges.
- **Standardization (1997)**:
  JavaScript was standardized by **ECMA International**, with the release of the first **ECMAScript (ES1)** specification to ensure consistency across browsers
- **AJAX Revolution (2000s)**:
  The introduction of **AJAX** allowed for dynamic, interactive web applications, boosting JavaScript's importance.
- **Frameworks and Libraries (2006–2010)**:
  Tools like **jQuery**, **AngularJS**, and **React** simplified development and made JavaScript more powerful.
- **Node.js (2009)**:
  JavaScript expanded to server-side programming with Node.js, enabling full-stack JavaScript development.
- **Modern Era (2015–Present)**:
  Regular ECMAScript updates (e.g., ES6 in 2015) introduced significant language improvements, while frameworks like **React** and **Vue.js** dominate modern web development.

Benefits of JavaScript

- **Lightweight**: JavaScript is designed to execute quickly and efficiently, making it ideal for client-side scripting.
- **Interpreted**: Unlike compiled languages, JavaScript code is executed line-by-line by the browser's JavaScript engine (e.g., V8 in Chrome, SpiderMonkey in Firefox).
- **Cross-platform**: JavaScript runs seamlessly on all major browsers and platforms.
- **Community Support**: JavaScript has a massive developer community and extensive libraries for solving common programming challenges.
- **Versatility**: It can be used across the full stack of web development, from client-side interactivity to server-side operations.
- **Integration**: Works seamlessly with other web technologies, including HTML, CSS, and third-party APIs.

Features

1. **Scripting for Web Pages**:
    1. JavaScript enables developers to add interactive features to web pages, such as buttons, forms, image sliders, and modal windows.
    2. It allows seamless integration with HTML and CSS to create a cohesive user experience.
2. **Dynamic Content Creation**:
    1. JavaScript can manipulate the Document Object Model (DOM) to dynamically update content on a webpage.
    2. Examples include updating a shopping cart without reloading the page or displaying live weather updates.
3. **High Interactivity**:
    1. Enables real-time interactivity, such as form validation, live chat applications, or drag-and-drop functionality.
    2. JavaScript facilitates event-driven programming, where specific user actions trigger responses in the application.

Places to put JavaScript code

In HTML documents, you can include **JavaScript** code in different locations. Here are the common places to put JavaScript:

1. **Inside the <script> Tag in the <head> Section**
2. **Inside the <script> Tag at the End of the <body> Section**
3. **External JavaScript File**
4. **Inline JavaScript (in HTML Attributes)**

1. Inside the <script> Tag in the <head> Section

The <head> section is used to include JavaScript that should run before the content of the page is fully loaded.

This can be useful for functions or libraries that need to be available when the page is rendered.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript in Head</title>
  <script>
    alert("JavaScript in the head!");
  </script>
</head>
<body>
  <h1>Welcome to JavaScript Example</h1>
</body>
</html>
```

2. Inside the <script> Tag at the End of the <body> Section

It is often considered a good practice to place JavaScript just before the closing </body> tag.

This ensures that the browser loads all the HTML content before running the script, improving page load speed and reducing render-blocking.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript at the End</title>
</head>
<body>
  <h1>Welcome to JavaScript Example</h1>

  <script>
    alert("JavaScript at the end of body!");
  </script>
</body>
</html>
```

3. External JavaScript File

External JavaScript is often the most organized and maintainable way to add JavaScript.

The code is placed in a separate .js file, which is then linked to the HTML document.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>External JavaScript</title>
</head>
<body>
  <h1>Welcome to External JavaScript Example</h1>
  <script src="script.js"></script>
</body>
</html>
```

```
alert("JavaScript from an external file!");  //JavaScript File (script.js)
```

4. Inline JavaScript (in HTML Attributes)

You can also write JavaScript directly inside HTML element attributes like onclick, onmouseover, etc.

This method is often used for event handling.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inline JavaScript</title>
</head>
<body>
  <button onclick="alert('Button Clicked!')">Click Me</button>
</body>
</html>
```

JavaScript comments

- JavaScript comments are used to add notes, explanations, or temporarily disable parts of the code.
- They are ignored by the JavaScript engine during execution.
- There are two types of comments in JavaScript: **single-line comments** and **multi-line comments**.

1. Single-Line Comments

- Start with //.
- Used for short notes or to disable a single line of code.

```
// This is a single-line comment
let name = "Alice"; // Variable declaration
console.log(name); // Output: Alice
```

2. Multi-Line Comments

- Enclosed between /* and */.
- Used for longer explanations or to comment out multiple lines of code.

```
/*
  This is a multi-line comment.
  You can use it to explain complex code or disable multiple lines.
*/
let age = 25;
console.log(age); // Output: 25
```

Best Practices for Writing Comments

- Keep comments concise and meaningful.
- Avoid obvious comments (e.g., // Declare a variable for let x = 10;).
- Regularly update comments to match changes in code.
- Use comments to explain *why* something is done, not *what* the code does (the code itself should be clear enough).

JavaScript Variables

A variable in JavaScript is a container for storing data values. Variables allow developers to store, update, and manipulate data dynamically.

e.g. myVar, _count, $price

Rules of Naming Variables in JavaScript

When naming variables in JavaScript, follow these rules:	
Start with a letter, underscore (_), or dollar sign ($):	• Valid: myVar, _count, $price • Invalid: 1var (starts with a number)
Use letters, digits, underscores, or dollar signs after the first character:	• Valid: var123, _name1, $total • Invalid: my-var (hyphen is not allowed)
Case-sensitive:	• myVar and myvar are different variables.
Reserved keywords cannot be used as variable names:	• Invalid: let, class, function, return
Descriptive names are recommended:	• Use names that convey meaning, such as userName or totalAmount.

Declaring Variables

JavaScript provides three keywords to declare variables:

- var,
- let,
- const.

1. var (Old way, avoid using in modern development)

- Function-scoped.
- Can be re-declared and updated.

```
var name = "Alice";
console.log(name); // Output: Alice
```

2. let (Modern and recommended for mutable variables)

- Block-scoped.
- Can be updated but not re-declared in the same scope.

```
let age = 25;
age = 30; // Allowed
console.log(age); // Output: 30
```

3. const (Modern and recommended for constants)

- Block-scoped.
- Must be initialized during declaration and cannot be updated or re-declared.

```
const pi = 3.14;
// pi = 3.15; // Error: Assignment to constant variable
console.log(pi); // Output: 3.14
```

Types of Variables

JavaScript variables can hold different types of data, which are broadly categorized into

primitive types and

non-primitive (reference) types.

1. Primitive Types:

- **Number**: Represents numeric values,

let age = 25; // Number

- **String**: Represents text,

let name = "Alice"; // String

- **Boolean**: Represents true or false,

let isActive = true; // Boolean

- **Undefined**: A variable declared but not assigned a value,

let x; // Undefined

- **Null**: Represents an explicitly empty or non-existent value,

let data = null; // Null

- **Symbol**: Represents a unique identifier (introduced in ES6),

let id = Symbol("id"); // Symbol

- **BigInt**: Used for very large numbers (introduced in ES11),

let bigNumber = 1234567890123456789012345678901234567890n; // BigInt

2. Non-Primitive (Reference) Types:

- **Object**: Collection of key-value pairs.

```
let user = { name: "Alice", age: 25 };
```

- **Array**: An ordered list of values

```
let colors = ["red", "green", "blue"];
```

- **Function**: A block of code designed to perform a particular task.

```
function greet() {
  return "Hello!";
}
```

Examples of Declaring and Using Variables

Example 1: Primitive Types

```
let name = "John";      // String
let age = 30;           // Number
let isLoggedIn = true;  // Boolean
let salary;             // Undefined
let emptyValue = null;  // Null
```

Example 2: Non-Primitive Types

- let person = { name: "Alice", age: 25 }; // Object
- let fruits = ["apple", "banana", "cherry"]; // Array

Summary of Data Types

Type	Description	Example
Number	Numeric values	42, 3.14
String	Text	"Hello", 'World'
Boolean	Logical values	true, false
Undefined	Uninitialized variables	let x;
Null	Empty or non-existent value	null
Symbol	Unique and immutable identifier	Symbol("id")
BigInt	Large integers	12345678901234567890n
Object	Collection of key-value pairs	{ name: "John" }
Array	Ordered list of values	["red", "blue"]
Function	Block of reusable code	function greet() {}

Types of Variables in JavaScript

In JavaScript, variables can have different scopes, which define where the variables are accessible within the code.

Two main types of variables based on scope are **global variables** and **local variables**.

1. Global Variables

Definition:
Global variables are declared outside any function, block, or method and are accessible throughout the entire script, including inside functions or blocks.

Characteristics:

- Accessible from any part of the program.
- They persist in memory as long as the program runs.
- Can cause unexpected behavior if not used carefully, especially in large programs.

Example:

```
let globalVar = "I am a global variable"; // Declared outside any function


function showGlobal() {
   console.log(globalVar);
// Accessible here
}


showGlobal();
// Output: I am a global variable
console.log(globalVar);
// Output: I am a global variable
```

Key Points:

- Avoid creating global variables unnecessarily as they can lead to conflicts in larger applications.
- Variables declared without let, const, or var automatically become global (not recommended).

2. Local Variables

Definition:
Local variables are declared inside a function, block, or method and are only accessible within that scope.

Characteristics:

- Created when the function or block is executed.
- Destroyed after the function or block completes execution.
- Not accessible outside their scope.

Example

```
function showLocal() {
    let localVar = "I am a local variable"; // Declared inside a function
    console.log(localVar); // Accessible here
}

showLocal(); // Output: I am a local variable
// console.log(localVar); // Error: localVar is not defined
```

Block Scope (let and const):

- Variables declared with let or const inside a block {} are only accessible within that block.

```
{
   let blockVar = "I exist only in this block";
   console.log(blockVar); // Output: I exist only in this block
}
// console.log(blockVar); // Error: blockVar is not defined
```

Differences Between Global and Local Variables

Feature	Global Variables	Local Variables
Scope	Entire script or program	Only inside the function or block where defined
Lifetime	Exists as long as the program is running	Created when the function/block is executed; destroyed afterward
Accessibility	Accessible from any part of the program	Not accessible outside the function/block
Declaration	Declared outside any function/block	Declared inside a function/block
Memory Usage	Can use more memory if not managed carefully	Memory is released when function/block ends