UNIT IV

(Client Side Scripting using JAVA SCRIPT)

JavaScript is a **high-level, interpreted programming language** used to create **interactive and dynamic web pages**. It is one of the core technologies of web development along with HTML and CSS. JavaScript helps in adding functionality such as form validation, animations, and user interaction to websites.
It is a **client-side scripting language**, meaning it runs directly in the user's web browser without needing a server. It is widely used in modern web development and also supports **object-oriented programming concepts**.

History of Javascript:-

Created by Brendan Eich developed in 1995 at Netscape. Initially called mocha, later Livescript finally Javascript.

Role of JavaScript in Web Development

JavaScript is a **high-level, interpreted, and object-oriented programming language** used to create **interactive and dynamic web pages**. It is one of the three core technologies of web development along with HTML and CSS. JavaScript runs directly in the web browser and helps in adding behavior to web pages.

1. Adding Interactivity

JavaScript is mainly used to make web pages interactive and responsive to user actions. It allows users to perform actions like clicking buttons, submitting forms, and navigating pages smoothly. It handles events such as mouse clicks, keyboard input, and scrolling.

2. Client-Side Validation

JavaScript is used to validate user input before sending it to the server. It checks whether the entered data is correct and complete. For example, it can validate email formats, passwords, and required fields. This reduces errors and improves data accuracy.

3. Dynamic Content Updates

JavaScript allows web pages to update content dynamically without reloading the page. It uses the Document Object Model (DOM) to modify elements like text, images, and styles. This makes websites faster and more efficient.

4. Improving User Experience

JavaScript plays a key role in enhancing the overall user experience. It enables animations, transitions, and visual effects on web pages. Features like image sliders, modals, and tooltips are created using JavaScript. These elements make websites more attractive and engaging.

5. Backend Development

JavaScript is not limited to frontend development; it is also used for backend programming. With technologies like Node.js, developers can build server-side applications. It allows handling databases, server requests, and APIs.

6. Integration with APIs

JavaScript is used to connect web applications with external services using APIs. It can fetch and display data from servers dynamically. For example, it can show weather updates, maps, or live data feeds. APIs help in integrating third-party services into websites.

7. Building Web Applications

JavaScript is widely used to build modern web applications. It is used in developing applications like e-commerce sites, online banking, and social media platforms. Frameworks like React, Angular, and Vue are based on JavaScript. These frameworks help in building complex applications easily.

Ways to Add JavaScript in HTML

JavaScript can be added to an HTML document in different ways to make web pages interactive and dynamic. The three main methods are Inline JavaScript, Internal JavaScript, and External JavaScript.

1. Inline JavaScript

Inline JavaScript is written directly inside HTML elements using attributes like onclick, onmouseover, etc. It is the simplest way to add JavaScript for small tasks. This method is mainly used for quick actions such as button clicks.

Ex:-

```
<body>
  <h2>inline javascript Example</h2>
  <button onclick="alert('shri gnanambica degree college')">Click me</button>
</body>
```

2. Internal JavaScript

Internal JavaScript is written inside the <script> tag within the HTML file. It is usually placed in the <head> or at the end of the <body> section. This method is useful when the script is specific to a single webpage.

Ex:-

```
<body>
  <script>
    document.write("Shri Gnanambica")
  </script>
</body>
```

3. External JavaScript

External JavaScript is written in a separate file with a .js extension. It is linked to the HTML file using the <script src="filename.js"></script> tag. This method is best for large projects and professional development. It keeps HTML and JavaScript completely separate, improving maintainability.

Step 1: Create HTML file

```
<html>
  <head>
    <title>Example of external javascript</title>
    <script src = "script.js"></script>
  </head>
  <body>
```

```
    </body>
  </html>
```

Step 2: Create JavaScript file (script.js)
JavaScript is written in a separate .js file.

```
document.write('Hello, World!');
```

Displaying output:-

1. using console.log()

Console.log() prints the result in the browser console.

Ex:-

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    console.log("Shri Gnanambica Degree College");
    console.log("madanapalle")
  </script>
</body>
</html>
```

How to see output:-

1. open browser
2. right click→ inspect
3. click console.

2). Using document.write()

document.write() prints output directly on the webpage.

Ex:-

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
```

```
</head>
<body>
  <script>
    document.write("Shri Gnanambica Degree College");
    document.write("<br>");
    document.write("madanapalle");
  </script>
</body>
</html>
```

In this example :
Document.write → function
; (semicolon)→ statement end

3. using alert() :-
alert() displays output in a popup box.

Ex:-

```
<!DOCTYPE html>
<html>
<body>
<script>
alert("Welcome to JavaScript");
</script>
</body>
</html>
```

History of JavaScript

1. Beginning of JavaScript (1995)

JavaScript was created in 1995 by Brendan Eich at Netscape.
It was initially called Mocha, then renamed LiveScript.
Finally, it was named JavaScript for popularity.
It was developed to make web pages interactive.
Mainly used for client-side scripting.

2. Standardization (1997)

JavaScript was standardized by ECMA International in 1997.
The standard version is called ECMAScript (ES).
First version ES1 was released in 1997.

3. Early Versions (1997–1999)

ES1 (1997) was the first official version.
ES2 (1998) included minor updates.
ES3 (1999) introduced major improvements.
Features like error handling and regular expressions were added.
These versions formed the base of JavaScript.

4. ES5 (2009)

ES5 was released in 2009 with important updates.
Introduced strict mode for better error handling.
Added JSON support for data exchange.
New array methods like map(), filter() were included.
Improved browser compatibility and performance.

5. ES6 / ECMAScript 2015

ES6 was released in 2015 with major improvements.
Introduced let and const for variables.
Added arrow functions and classes.
Template literals improved string handling.
Made JavaScript modern and powerful.

Variables in JavaScript

A variable in JavaScript is used to store data values. It acts as a container to hold information that can be used later. Variables can store different types of data like numbers, strings and objects.

In JavaScript variables are declared using var, let and const.

Rules of Naming Variables in JavaScript:-

There are some rules while declaring a javascript variable (also known as identifiers).

- Name must start with a letter (a to z or A to Z), underscore(_) or dollar ($) sign.
- After the first letter, we can use digits(0 to 9), for example , value1.
- JavaScript variables are case-sensitive for example a and A are different variables.

Types of Variables keywords:-

1. var keyword

The var keyword in JavaScript is used to declare variables. It was the old method of creating variables before the introduction of let and const.
It allows re-declaration and has function scope, which can cause problems in large programs.

Ex:- (can be re-declared)

```
var a = 10;
var a = 20;   // allowed
console.log(a);
```

Ex:- (can be updated)

```
var a = 10;
a = 50;
document.write(a);
```

2. let keyword

The let keyword in javascript is used to declare variables. It was introduced in ES6(ECMAScript 2015) as an improved version of var.
A variable declared with let can store data values and its value **can be updated**, but it **cannot be re-declared** in the same scope.

Ex:-

```
let a = 10;
let name = "Gnanambica";
document.write(a);
document.write(“<br>”);
document.write(name);
```

Ex:- (cannot be re-declared)

```
let a = 10;
let a = 20;              o/p:- error
document.write(a);
```

Ex:- (can be updated)

```
let a = 10;
a = 50;     //  Allowed
document.write(a);
```

Using const keyword:-

The const keyword in javascript is used to declare variables whose values **cannot be changed** after initialization. It was introduced in ES6 (ECMAScript 2015).
A variable declared with const must be assigned a value at the time of declaration and **cannot be reassigned** later.

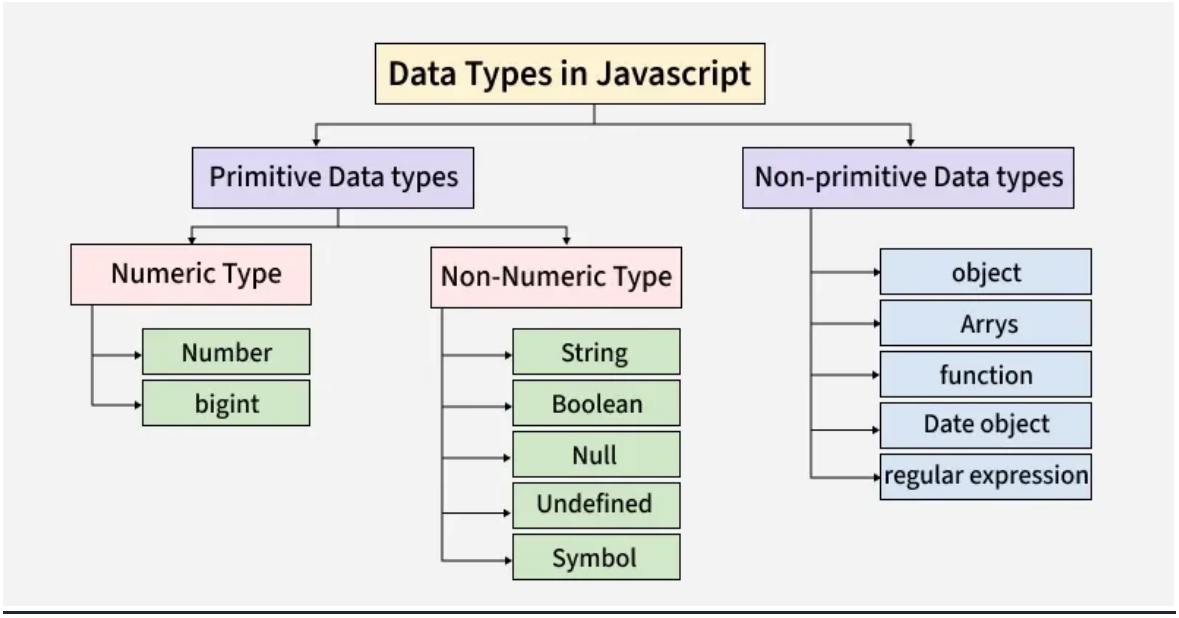
Ex:-

```
const num = 3.14;
const name = "Gnanambica";
```

Ex:- (cannot be re-declared)

```
const a = 10;
const a = 20;  //  Error
```

Data Types in JavaScript:-



Data types in JavaScript define the type of data a variable can hold. JavaScript is a dynamically typed language, which means you don't need to specify the data type explicitly. The type is automatically determined when a value is assigned.
JavaScript data types mainly divided into primitive and Non-primitive types.

1. Primitive Data Types:-

Primitive data types store single values and are immutable. They are simple and directly stored in memory.

1. Number

The Number type represents both integer and floating point values. It includes positive, negative and decimal numbers.

Ex:-

```
let a = 10;
let b = 3.14;
document.write(a);
document.write(b);
document.write(typeof a);
```

2. String

String is used to represent text data. It is written inside single(‘ ‘), double (“ “) or backticks(` `).

Strings can include letters, numbers and symbols. They are immutable in JavaScript.

Ex:-

```
let name = "Rama";
console.log(name);
```

3. Boolean

Boolean is a primitive data type in JavaScript that stores only two values: true or false. It is mainly used in decision-making statements such as if , while and comparisons.

Ex:-

```
let a = 10;
let b = 5;
console.log(a > b);
console.log(a < b);
```

Ex:-

```
let isStudent = true;
let isAdmin = false;

console.log(isStudent);   // true
console.log(isAdmin);     // false
```

4. Undefined

A variable that is declared but not assigned a value has the type undefined.

It means the variable exists but has no value.

It is automatically assigned by JavaScript.

Represents absence of value.

Ex:-
let x;
console.log(x); // undefined

5. Null

null represents an intentional absence of value.
It is assigned by the programmer.
It is different from undefined.
Used when a variable should have no value.
Ex:-
let age = null;
console.log(age);

6. BigInt

BigInt is used to store very large integers.
It is used when numbers exceed the safe limit of Number type.
It is created by adding n at the end.
Useful for high-precision calculations.
Example:
let big = 12345678901234567890n;

7. Symbol

symbol is a primitive data type introduced in ES6(ECMAScript 2015). It is used to create a unique identifier for variables or object properties. Every symbol value is unique, even if two symbols have the same description. A symbol is created using the Symbol() function. The value inside the parentheses is only a description and not the actual value.
Ex:-
let id1 = Symbol("user");
let id2 = Symbol("user");

console.log(id1 === id2); // false

Non-Primitive Data Types

Non-primitive data types in JavaScript are used to store **multiple values or complex data**. Non-primitive values are mutable, which means they can be changed after creation.

1. Object

An object is a collection of key-value pairs. Each property in an object has a name and a value. Objects are used to represent real-world entities. Values inside objects can be of any data type.

Ex:-

```
let student = {
   name: "Vivekananda",
   age: 22,
   course: "MSc"
};

console.log(student.name);
```

2. Array

An array stores multiple values in a single variable. Each value is stored with an index number starting from 0 Arrays can store numbers, strings, or mixed values.

Ex:-

```
let fruits = ["Apple", "Mango", "Banana"];
console.log(fruits[1]);
```

3. Function

A function is a reusable block of code.
Functions can be stored in variables.
They can accept inputs and return outputs.
Functions help avoid repeated code.
JavaScript treats functions as objects.

Example

```
function  greet() {
   console.log("Hello");
}

greet();
```

4. Date

The Date object is used to work with dates and time.
It can display current date and time.
Date methods help get day, month, and year.
Useful in calendars and scheduling apps.
It simplifies time-related tasks.

Example

```
let today = new Date();
console.log(today);
```

5. Regular Expression

Regular expressions are used for pattern matching.
They help search text in strings.
Used for validation like email or phone number.
They improve text processing.
JavaScript provides built-in RegExp support.

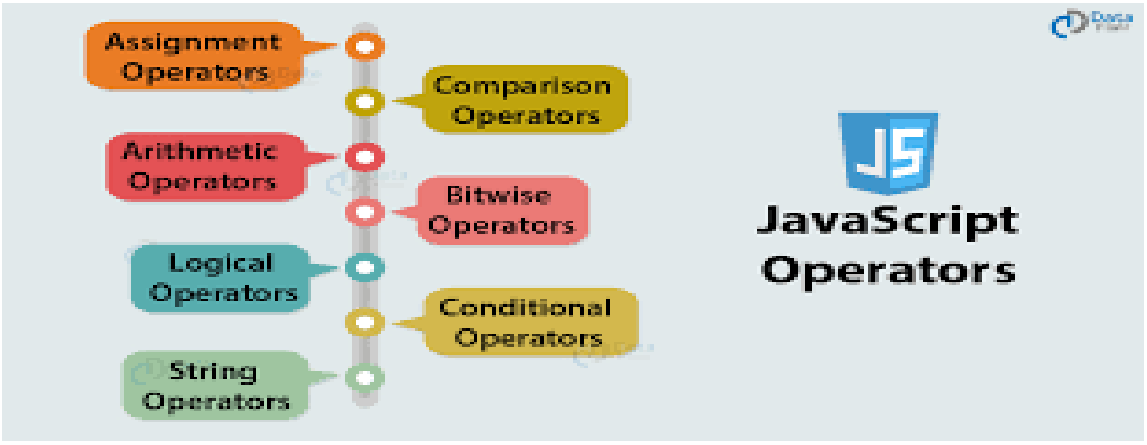
Example

```
let pattern = /java/i;/;

console.log(pattern.test("JavaScript"));
```

Operators in JavaScript

Operators in JavaScript are special symbols used to perform operations on variables and values. They help in calculations, comparisons, and logical decisions in a program. Operators make JavaScript programs more dynamic and useful.



1. Arithmetic Operators

Arithmetic operators are used to perform mathematical calculations. They work with numeric values. These operators include addition, subtraction, multiplication, and division. They are commonly used in calculations and they return numeric results.
Common arithmetic operators include addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).
Ex:-

```
let a = 10;
let b = 5;

console.log(a + b);
console.log(a - b);
console.log(a * b);
console.log(a / b);
```

2. Assignment Operators

Assignment operators are used to assign values to variables in JavaScript. The simple assignment operator (=) stores a value in a variable. JavaScript also provides combined assignment operators like +=, -=, *=, and /=. These operators perform an operation and assign the result back to the variable.

Ex:-

```
let x = 20;
console.log("Initial value:", x);
x += 5;
console.log("Addition :", x);
x -= 3;
console.log("Subtraction:", x);
x *= 2;
console.log("Multiplication:", x);
x /= 4;
console.log("Division :", x);
x %= 3;
console.log("Modulus :", x);
```

3. Comparison Operators

Comparison operators are used to compare two values in JavaScript. These operators return a Boolean value of true or false. Common comparison operators include equal to (==), strict equal (===), greater than (>), and less than (<). They are mainly used in conditional statements like if and loops.

Ex:-

```
let a = 10;
let b = 20;

console.log(a < b);
console.log(a == b);
```

4. Logical Operators

Logical operators are used to combine or reverse Boolean conditions. The main logical operators are AND (&&), OR (||), and NOT (!). They help in checking multiple conditions at the same time. Logical operators are mostly used in if statements and loops. They return either true or false depending on the condition.

Ex:-

```
let username="admin";
let password=1234;
   if (username=="admin" && password==1234){
     document.write("Login successfully...");
   } else {
     document.write("Login failed");
   }
```

(or)

Ex:-

```
let username = prompt("Enter username:");
let password = Number(prompt("Enter password:"));

if (username == "admin" && password == 1234) {
   document.write("Login successfully...");
} else {
   document.write("Login failed");
}
```

5. String Operators

String operators are used to combine text values. The + operator joins two strings together. This process is called concatenation. It helps display dynamic messages.

Ex:-

```
let first = "Hello";
let second = "World";

console.log(first + " " + second);
```

6. Bitwise Operators

Bitwise operators work on binary numbers. Common operators are &, |, and ^. They perform operations bit by bit. These operators are used in advanced programming.

1. Bitwise AND (&)
2. Bitwise OR (|)
3. Bitwise XOR (^)

Ex:- (Bitwise AND (&))

```
let a = 5;
let b = 3;
console.log(a & b);
```

Ex:- (Bitwise OR (|))

```
let a = 5;
let b = 3;
document.write(a | b);
```

Ex:- (Bitwise XOR (^))

```
let a = 5;
let b = 3;
document.write(a ^ b);
```

7. Conditional Operator

The conditional operator is a shortcut for if...else. It uses ? and : symbols. It checks a condition and returns one of two values. It makes code shorter. It improves readability.

Example

```
let age = 18;
let result = age >= 18 ? "Adult" : "Minor";

console.log(result);
```

Control Statements in JavaScript

Control statements in JavaScript are used to control the flow of program execution. They decide which statement should execute and how many times it should run. These statements help in decision-making and repetition of tasks. Control statements make programs more flexible and interactive.

They are mainly divided into :

1. decision-making statements
2. looping statements
3. branching statements.

Understanding control statements is important for writing logical programs.

Decision Making Statements

Decision making statements in JavaScript are used to choose different actions based on conditions. They allow the program to execute a specific block of code when a condition is true. They are mainly used in validations, login systems, and user-based actions.

1. if statement
2. if-else statement
3.else if ladder
4. switch statement

1. if Statement

The if statement executes a block of code only when a condition is true. If the condition is false, the block is skipped. It is useful when only one condition needs to be checked. The condition usually returns true or false.

Ex:-

```
let age = 20;

if (age >= 18) {
   document.write("Eligible to vote");
}
```

2. if- else Statement

The if...else statement is used when there are two possible outcomes. If the condition is true, the if block executes. If the condition is false, the else block executes. It helps choose between two actions.

Ex:-

```
let marks = 30;

if (marks >= 35) {
   document.write("Pass");
} else {
   document.write("Fail");
}
```

3. else if Ladder

The else if ladder is used to check multiple conditions one after another. It executes the first true condition block. If none of the conditions are true, the else block executes. It is useful when multiple choices are available. This statement avoids writing many separate if statements.

Ex:-

```
let marks = 85;

if (marks >= 90) {
   document.write("Grade A+");
} else if (marks >= 75) {
   document.write("Grade A");
} else {
```

```
   document.write("Grade B");
}
```

Ex:-

```
let day=Number(prompt("Enter number:"));
   if(day==1){
     document.write("Monday")
   }
   else if(day==2){
     document.write("Tuesday");
   }
   else if (day==3){
     document.write("Wednesday");
   }
   else{
     document.write("Invalid Day");
   }
```

4. switch Statement

The switch statement is used when one variable is compared with many values. Each possible value is written as a case. When a match is found, the corresponding code executes. The break statement stops further execution.

Ex:-

```
let day = 2;

switch(day) {
   case 1:
      document.write("Monday");
      break;
   case 2:
      document.write("Tuesday");
      break;
   default:
      document.write("Invalid");
}
```

Looping Statements

Looping statements in JavaScript are used to execute a block of code repeatedly. They help reduce code repetition and save programming time. Loops are useful when the same task needs to be performed many times.

1. for loop
2. while loop
3. do-while loop

1. for Loop

The for loop is used when the number of iterations is known in advance. It contains initialization, condition, and increment or decrement in one line. The loop continues until the condition becomes false. It is commonly used for counting and displaying values.

Ex:-

```
for (let i = 1; i <= 5; i++) {
   document.write(i + "<br>");
}
```

2. while Loop

The while loop executes as long as the condition is true. It checks the condition before executing the loop body. If the condition is false at the beginning, the loop will not run.

Ex:-

```
let i = 1;
while (i <= 5) {
   document.write(i + "<br>");
   i++;
}
```

3. do...while Loop

The do...while loop executes the code at least once. It checks the condition after executing the loop body. Even if the condition is false, the loop runs one time. It is useful when the code must execute at least once.

Ex:-

```
let i = 1;

do {
   document.write(i + "<br>");
   i++;
} while (i <= 5);
```

Branching Statements

Branching statements in JavaScript are used to change the normal flow of program execution. They allow the program to stop, skip, or transfer control to another part of the code.

1. break
2. continue
3. return

1. break Statement

The break statement is used to terminate a loop or switch statement immediately. When the break statement is executed, control moves to the next statement after the loop. It helps stop unnecessary execution.

Ex:-

```
for (let i = 1; i <= 5; i++) {
   if (i == 3) {
      break;
   }
   document.write(i + "<br>");
}
```

2. continue Statement

The continue statement is used to skip the current iteration of a loop. After skipping, the loop moves to the next iteration. It does not stop the loop completely. This statement is useful when some values need to be ignored.

Ex:-

```
for (let i = 1; i <= 5; i++) {
   if (i == 3) {
      continue;
   }
   document.write(i + "<br>");
}
```

3. return Statement

The return statement is used inside a function to stop execution and send a value back. Once return is executed, the function ends immediately. It can return any data type such as number or string. It is useful for getting results from functions.

Ex:-

```
function add(a, b) {
   return a + b;
}
document.write(add(5, 3));
```

Functions in JavaScript

A function in JavaScript is a reusable block of code that performs a specific task. It helps reduce code repetition and makes programs easier to manage. Functions can accept input values called parameters. They can also return a result after execution.

Function Declaration

A function declaration defines a named function using the function keyword. It can be called anywhere in the program after creation. The function name identifies the function.

Ex:-

```
function greet() {
   document.write("Hello JavaScript");
}

greet();
```

Function with Parameters

Parameters are values passed into a function when it is called. They allow the function to work with different data.

Ex:-

```
function add(a, b) {
   document.write(a + b);
}

add(10, 20);
```

Advantages of Using Functions in JavaScript

Functions are one of the most useful features in JavaScript. They allow a block of code to be written once and used many times. Functions make programs easier to manage and understand. They help divide large programs into smaller parts.

1. Code Reusability

Functions allow the same code to be reused whenever needed. Instead of writing the same code repeatedly, a function can be called multiple times. This reduces duplication in the program. Reusable code saves development time. It also makes the program shorter.

2. Easy Maintenance

Functions make programs easier to maintain. If a change is needed, it can be made in one function only. There is no need to modify the same code in many places. This reduces errors in the program.

3. Better Readability

Functions improve the readability of code. A meaningful function name explains the purpose of the code. This helps other programmers understand the program easily. Large programs become more organized. It becomes easier to identify each task

4. Reduces Complexity

Functions divide a large program into smaller manageable parts. Each function handles a specific task. This reduces the complexity of the program. Smaller sections are easier to understand.

5. Easier Testing

Functions make testing easier in JavaScript. Each function can be tested separately. Errors can be found quickly inside a specific function. This saves time in debugging.

6. Improves Modularity

Functions support modular programming. Different functions can be created for different tasks. Each module works independently. This makes the program well-structured.

7.Easy Debugging

Errors can be found quickly inside functions. Only the specific function needs checking. This makes debugging easier. It saves troubleshooting time. It improves code quality. Problems can be isolated easily.

8.Supports Modularity

Functions support modular programming. Each module performs one operation. Modules can work independently. This improves organization. It helps teamwork in projects. It creates professional code.

Regular Expressions

Regular Expressions, also called RegEx, are special patterns used in JavaScript to search, match, and manipulate text. They help identify whether a string contains a specific sequence of characters. RegEx is written between forward slashes like /pattern/. They are mainly used for validating and searching text data. RegEx makes text processing easier and more efficient.

Ex:-

```
let pattern1 = /java/;
document.write(pattern1.test("javascript"));
```

(or)

```
let pattern2 = new RegExp("java");
document.write(pattern2.test("javascript"));
```

Application in Pattern Matching

Regular expressions are widely used for pattern matching in JavaScript. They can search for words, numbers, or symbols inside a string. Pattern matching helps locate required data quickly. It can identify whether a pattern exists in text. This is useful in searching documents and form data.

Ex:-

```
let text = "I am learning JavaScript";
let pattern = /JavaScript/;
document.write(pattern.test(text));
```

Application in Form Validation

RegEx is commonly used to validate user input in web forms. It can verify email addresses, mobile numbers, and passwords. This ensures data is entered in the correct format

Ex: (**Mobile Number Validation Example**)

Ex:-

```
let mobile = "9876543210";
let pattern = /^[0-9]{10}$/;
document.write(pattern.test(mobile));
```

Explanation

^ → start
[0-9] → digits only
{10} → exactly 10 digits
$ → end

(Or)

```
let mobile = prompt("Enter mobile number:");
let pattern = /^[0-9]{10}$/;
if(pattern.test(mobile)) {
   document.write("Valid Mobile Number");
} else {
   document.write("Invalid Mobile Number");
}
```

Symbol	Meaning	Example
^	Start of string	/^A/
$	End of string	/A$/
[ ]	Character range	/[0-9]/
+	One or more	/a+/
*	Zero or more	/a*/
{ }	Number of times	/[0-9] {10}/

test() method

Ex:- pattern.test(string)

Checks whether the given string matches the pattern and returns true or false.

Ex:- (**Name Validation Example)**

```
let name = prompt("Enter your name:");
let pattern = /^[A-Za-z ]+$/;

if(pattern.test(name)) {
   document.write("Valid Name");
} else {
   document.write("Invalid Name");
}
```

Common Validation Patterns

Field	Regular Expression
Name	/^[A-Za-z]+$/
Email	/^[a-z0-9]+@[a-z]+\.[a-z]{2,3}$/
Mobile	/^[0-9]{10}$/
Password	/^[A-Za-z0-9@#$%]{6,12}$/

Application in Searching Multiple Matches

RegEx can find multiple matches in a string using flags. The g flag searches all occurrences. The i flag ignores uppercase and lowercase differences. This helps in advanced pattern matching.

Global Flag(g):-

The g flag stands for **global search**.

It tells JavaScript to search the entire string.

Without g, only the first match is returned.

With g, all matching values are found.

It is mainly used with match().

Ex:- (Find Multiple Words)

```
let text = "cat dog cat bird cat";
let result = text.match(/cat/g);

document.write(result);
```

Explanation

text.match(/cat/g)

- cat → word to search
- /g → search all matches
- match() → returns all matches
- Result is an array
- All "cat" words are found

Application in Replacing Text

Regular expressions can replace matching text with new text. The replace() method is used for this purpose. It helps modify strings automatically. This is useful in data cleaning. It saves time in text processing.

Ex:-

```
let text = "I like Java";
document.write(text.replace(/Java/, "Python"));
```

DOM in JavaScript:-

Document : Entire webpage
Object : <h1> element becomes an object
Model : browser stores it in a tree structure

DOM Stands for Document Object Model. It is a programming interface that represents an HTML document as a tree of objects. Every element in a webpage such as headings, paragraphs, images and buttons become an object in the DOM. javaScript uses the DOM to access, modify, add or remove HTML elements dynamically. The DOM acts as a bridge between HTML and JavaScript making modern web development possible.

DOM methods in Javascript:-

DOM methods are special functions used by JavaScript to access and manipulate HTML elements. These methods help programmers find, modify, create, and remove elements from a webpage. The browser provides these methods through the document object.

1. getElementById()

The getElementById() method selects a single HTML element by its unique id. Every id in HTML should be different. This method returns the matching element object. After selecting the element, JavaScript can change its content or style.

Ex:-

```
<h1 id="title">Hello</h1>

<script>
document.getElementById("title").innerHTML = "Welcome";
</script>
```

Explanation:-

Document refers to the webpage
getElementById() finds the element
“title” is the id name
innerHTML chages the text

2. getElementsByTagName()

The getElementsByTagName() method selects all elements with the same tag name. It returns a collection of elements. Each element can be accessed using an index number. This method is useful when multiple elements have the same tag.

Ex:-

```
<p>First</p>
<p>Second</p>

<script>
document.getElementsByTagName("p")[1].innerHTML = "Changed";
</script>
```

Explanation

- Selects all <p> elements
- [1] means second paragraph
- innerHTML changes text
- Second paragraph becomes **Changed**
- Index starts from 0

3. getElementsByClassName()

The getElementsByClassName() method selects elements using class names. It returns all matching elements. Multiple elements can share the same class. It is useful for selecting groups of elements. The result can be accessed using index numbers.

Ex:-

```
<p class="demo">First</p>
<p class="demo">Second</p>

<script>
document.getElementsByClassName("demo")[0].innerHTML = "Updated";
</script>
```

Event Handling in JavaScript

Event handling in JavaScript is the process of responding to user actions on a webpage. An event can occur when a user clicks a button, types in a textbox, moves the mouse, or submits a form. JavaScript can detect these actions and execute a function. Event handling makes webpages interactive. It improves user experience.

Examples of Events

- onclick
- onmouseover
- onkeydown
- onchange
- onsubmit
- onload

1. onclick Event

The onclick event occurs when the user clicks an HTML element. It is commonly used with buttons and links. When the element is clicked, JavaScript executes a function.

Ex:-

```
<button onclick="showMessage()">Click Me</button>

<script>
function showMessage() {
   alert("Button Clicked");
}
</script>
```

Explanation

- A button is created
- onclick waits for click
- User clicks the button
- showMessage() runs
- Alert box appears

2. onmouseover Event

The onmouseover event occurs when the mouse pointer moves over an element. It is useful for changing styles or displaying messages.

Ex:-

```
<p onmouseover="changeText()">Move mouse here</p>

<script>
function changeText() {
   document.write("Mouse moved over paragraph");
}
</script>
```

Explanation

- Paragraph is displayed
- Mouse moves over it
- changeText() runs
- Message is displayed
- Action happens automatically

3. onkeydown Event

The onkeydown event occurs when the user presses a keyboard key. It detects key input immediately. It is useful in games and forms. It helps capture user actions. It works before the key appears in input.

Example

```
<input type="text" onkeydown="showKey()">

<script>
function showKey() {
   alert("Key Pressed");
}
</script>
```

Explanation

- Input field is created
- User presses a key
- onkeydown detects it
- Function runs
- Alert message appears

4. onchange Event

The onchange event occurs when the value of an input field changes. It works after the user changes the value and leaves the field. It is useful in forms. It helps validate data. It can update information dynamically.

Example

```
<input type="text" onchange="display()">

<script>
function display() {
  alert("Value Changed");
}
</script>
```

Explanation

- User enters text
- Value is changed
- User leaves field
- Function executes
- Alert is shown

5. onsubmit Event

The onsubmit event occurs when a form is submitted. It is used to validate form data before sending. It can stop submission if data is invalid. It is important in web forms. It improves data accuracy.

Example

```
<form onsubmit="return validateForm()">
  <input type="submit">
</form>

<script>
function validateForm() {
  alert("Form Submitted");
  return true;
```

```
}
</script>
```

Explanation

- Form is created
- User clicks submit
- validateForm() runs
- Alert message appears
- Form submits after validation

6. onload Event

The onload event occurs when the webpage finishes loading. It runs automatically. It is used to display messages or initialize data. It ensures the page is ready. It is useful for startup tasks.

Example

```
<body onload="welcome()">

<script>
function welcome() {
   alert("Page Loaded Successfully");
}
</script>
</body>
```

Explanation

- Page opens in browser
- Browser finishes loading
- welcome() runs
- Alert appears automatically
- User sees welcome message

Event	Occurs When
`onclick`	User clicks element
`onmouseover`	Mouse moves over element
`onkeydown`	Keyboard key pressed
`onchange`	Input value changed
`onsubmit`	Form submitted
`onload`	Page fully loaded

5 Marks:

1. what is JavaScript?
2. Define variables in javascript
3. List arithmetic operators in javascript.
4. Define the DOM (document object model)
5. What is a function in JavaScript?

10 Marks :

1. Define JavaScript and explain its role in web development.

2. Discuss the different datatypes in JavaScript with suitable examples.

3. Explain JavaScript variables and operators, highlighting their types and uses.

4. Describe functions in JavaScript, their syntax and advantages of using functions.

5. Analyze the role of control statements in javascript with examples of decision-making and looping structures.

6. Define Regular Expressions(RegEx) in javascript and discuss their applications in pattern matching.

B. Devendra Msc-CS
Dept of Computer Science & Applications
Shri Gnanambica Degree College(A).