**JavaScript**

**JavaScript Introduction:-**

**Ans:-(1)**

Javascript is most popluar scripting programming language. Javascript is client & server both side used. which is used to created web side, game etc…using dynamic content.

**Role of web devlopment:-**

🡪Using javascript is adds interactivity to web pages. It's used to create dynamic content, animations, and more.

🡪JavaScript can dynamically update HTML and CSS to change the content of a web page.

🡪 JavaScript can take data entered into a form and use it to generate a response.

**Ans:-(2)**

**How is JavaScript different from other programming language:-**

**1) Runs in the Browser** :- JavaScript is primarily used to run inside web browsers, making websites interactive.

2) **Interpreted, Not Compiled**:- JavaScript doesn’t need a separate compilation. It runs directly in the browser or runtime environment.

3) **Object-Based:-** While JavaScript supports Object-Oriented Programming, it class-based structures like Java or C#.

**Ans:-(3)**

**Discuss the use of <script>tag in HTML .how can you link external Javascript in HTML:-**

The <script> tag in HTML is used to embed JavaScript code within a web page.

Example:-

<script>

alert("Hello, World!");

</script>

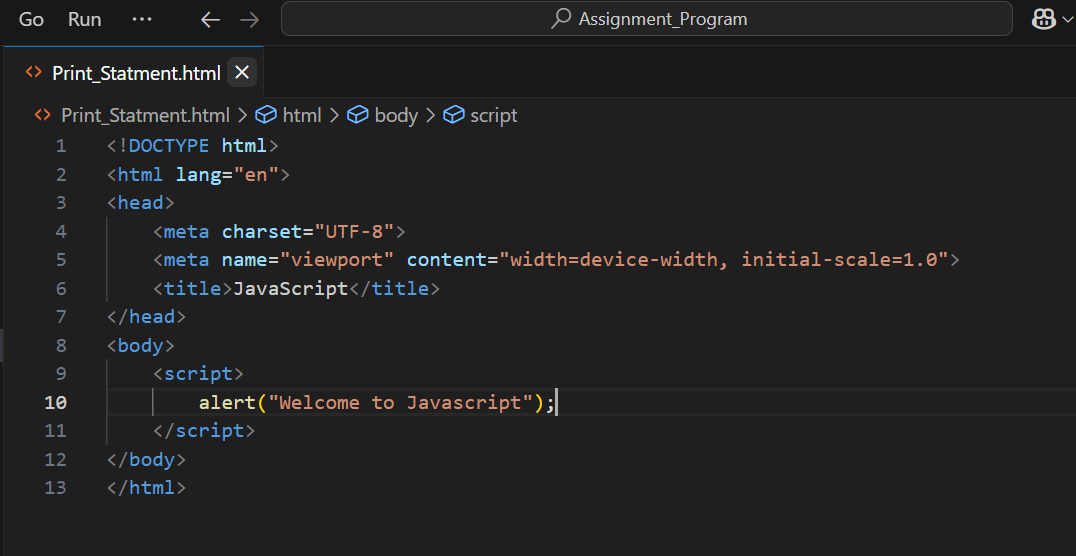
**External Javascript in HTML:-**

JavaScript in an external file and link into Use the <script> tag with the src attribute to reference an external .js file.

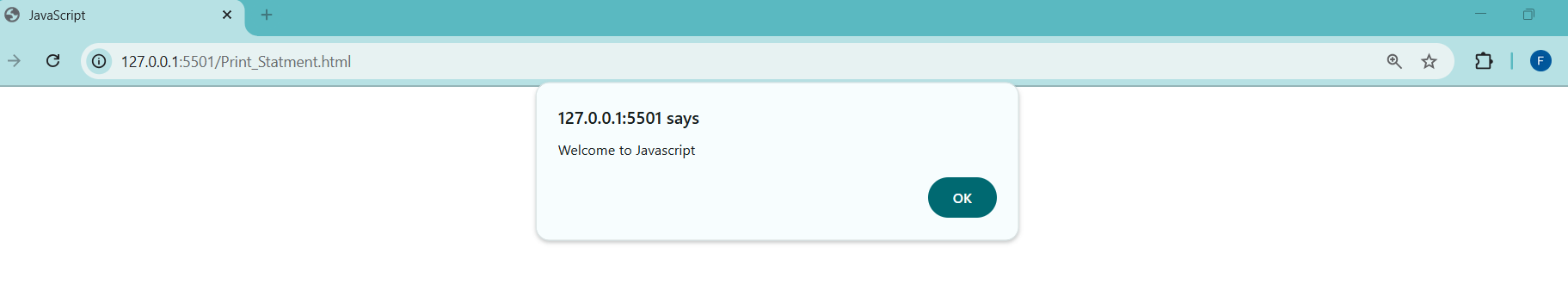
Example:-

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

Lab Assisment:-

****

|  |
| --- |
| **Output:-** |

****

**Variables and Data Types:-**

**Ans(1):-What are variables in JavaScript? How do you declare a variable:-**

Variable are container for storing value.In Javascript 3 diffrente way to declare variable.

**Let:-** This is can’t redeclare but reassign is possible using let keywords.

**Var:-** Redeclare and reassign both are possible using var keyword.

**Const:-** Redeclare and reassign both are not possible using const keywords.

Example:-

let a=23;

console.log(a)

var name=’Foram’;

console.log(name);

const pi=3.14;

console.log(3.14)

**Ans(2):- Explain the different data types in JavaScript. Provide examples for each.**

Javascript is mainly 2 categories .Primitive and Non-primitive data-type.

**Primitive :-** Primitive data type that means only store single value and changeable it.

(1)Number:-integer number.

Example:-

|  |
| --- |
| let age = 25;  let price = 99.99;  console.log(age, price);  Output: 25 99.99 |

(2)String:- string is collection of characters.

Example:-

|  |
| --- |
| let name = "John Doe";  console.log(name);  Output: string |

(3)Boolean:- Boolean is store true or flase value. true representation1 and false representation 0.

Example:-

|  |
| --- |
| **let isLoggedIn = true;**  **console.log(isLoggedIn);**  **Output: Boolean** |

(4)Undefine:- undefined means variable that has not been assigned a value is of type.

Example:-

|  |
| --- |
| **let x;**  **console.log(x);**  **Output: undefined** |

(5)Null :- Null is a special value that represents an empty or unknown value.

Example:-

|  |
| --- |
| **let y = null;**  **console.log(typeof y);**  **Output: object** |

**Non-Primitive:-** Non-primitive data type that means store multiple value and that can’t be changeable.

(1)Object:- Object is key value pairs.

Example:-

|  |
| --- |
| **let person = { firstName: "John", lastName: "Doe", age: 30 };**  **console.log(person);**  **Output: john , Doe,30** |

(2)Array:-Array is collection of similar data type.

Example:-

|  |
| --- |
| **let numbers = [1, 2, 3, 4, 5];**  **console.log(numbers);**  **Output: 1,2,3,4,5** |

**Ans(3) :- What is the difference between undefined null in JavaScript:-**

Undefined:- A variable that has been declare but has not been assigned value.

Example:-

let x;

console.log(x);

**Output:-**

Undefined

Null:- Represents an intentional absence value. type is object.

Example:-

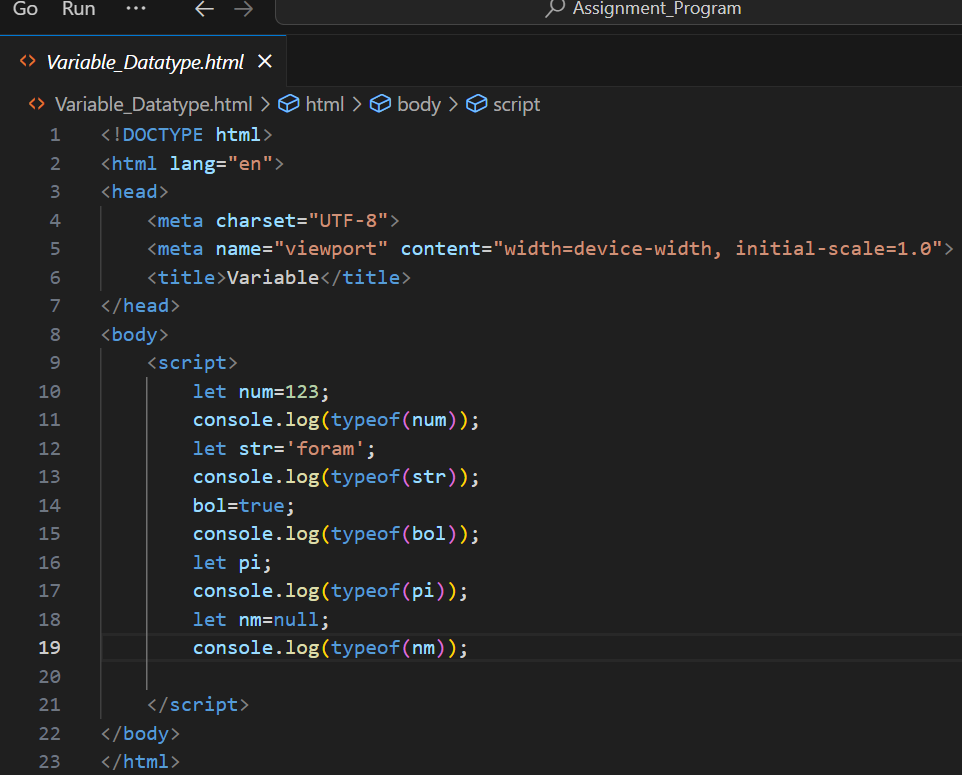
let y = null;

console.log(y);

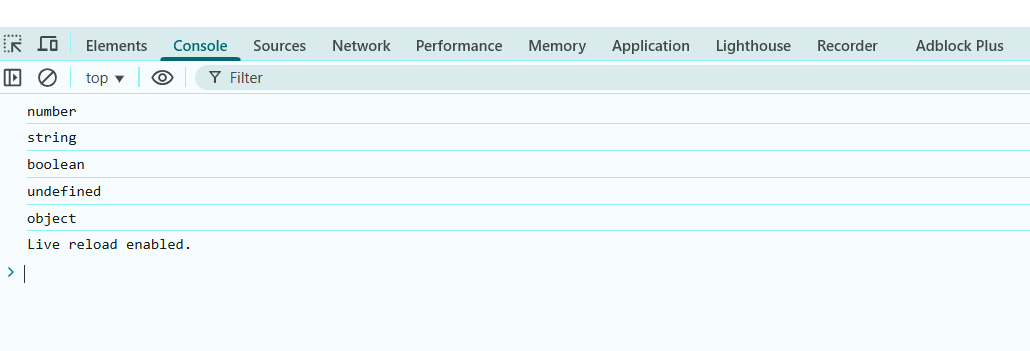
**Output:-**

Null

**Lab Assignment:-**

****

**Output:-**



**Operator:-**

**Ans(1):- Different types of javascript Operator:-**

🡪Arithmetic Operators

🡪Assignment Operators

🡪Comparison Operators (Relational Operators)

🡪Logical Operators

🡪Ternary Operators

🡪Type of Operators

🡪String Operators

Examples:

* **Arithmetic Operators:** (+,-,\*,/,%,\*\*,++,--)

let sum = 10 + 5; // 15

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| + | Addition | 5 + 3 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| - | Subtraction | 10 - 7 | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| \* | Multiplication | 4 \* 2 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| / | Division | 20 / 4 | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| % | Modulus (remainder) | 10 % 3 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| \*\* | Exponentiation | 2 \*\* 3 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| ++ | Increment | let x = 5; x++; | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| -- | Decrement | let y = 3; y--; | 2 |

**Assignment Operators:** (+=,-=,\*=,/=,%=,\*\*=)

let x = 10;

x += 5; // x = 15

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| = | Assign | x=10 | 10 |
| += | Add and assign | x+=5 (x=x+5) | 15 |
| -= | Subtract and assign | x-=2 (x=x-2) | 8 |
| \*= | Multiply and assign | x\*=3 (x=x\*3) | 30 |
| /= | Divide and assign | x/=2 (x=x/2) | 5 |
| %= | Modulus and assign | x%=3 (x=x% 3) | 1 |

**Comparision Operators:** (<,>,<=,>=,==,===,!=,!==)

console.log(10 > 5); // true

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| == | Equal to | 5 == "5" | True |
| === | Strictly equal | 5 === "5" | False |
| != | Not equal | 5 != "6" | True |
| !== | Strictly not equal | 5 !== "5" | True |
| > | Greater than | 10 > 5 | True |
| < | Less than | 5 < 10 | True |
| >= | Greater than or equal | 10 >= 10 | True |
| <= | Less than or equal | 5 <= 10 | True |

**Logical Operators:** (&&,||,!)

console.log(true && false); // false

| **Operator** | **Description** | **Example** | **Result** |
| --- | --- | --- | --- |
| && | Logical AND | true && false | false |
| || | Logical OR | || |  |
| ! | Logical NOT | !true | false |

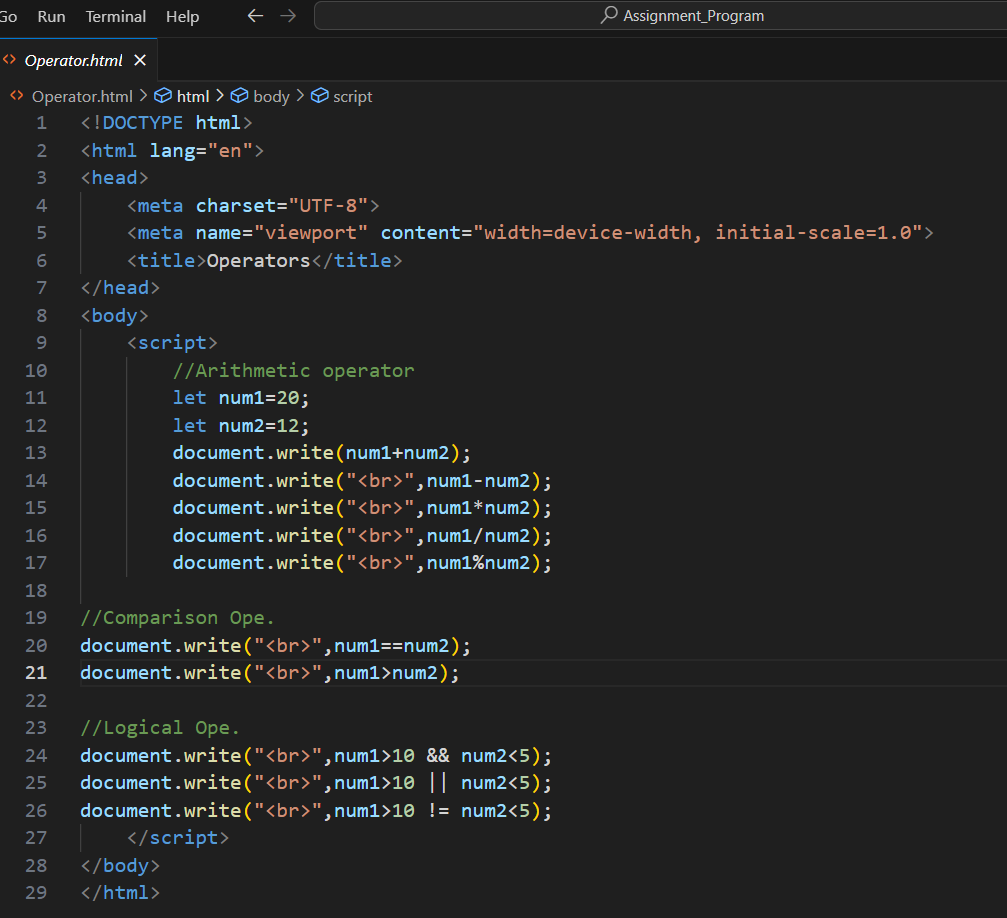
**Ans(2):- Between == and === in JS:-**

* ==(Equality):- Check only Value
* ===(Strict Equality):- Check Value and Data type

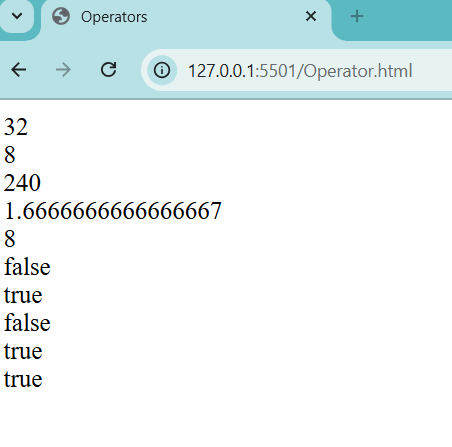
The == Operators Compares the value of two variables after performing type conversion.

The === Operators Compares the value of two variables Without performing type conversion.

**Lab Assignment:-**

****

**Output:-**

****

**Control Flow (If-Else, Switch):-**

**Ans(1):-**

**What is control flow in JS:-**

Control flow is the order in which JavaScript executes your code. It helps the program decide what to do next based on conditions or logic.

**How if-else Works:** An **if-else** statement checks a condition. If the condition is **true**, it runs one block of code. If it’s **false**, it runs another block.

**Example:**

let age = 20;

if (age >= 18) {

console.log("You are eligible to vote.");

} else {

console.log("You are not eligible to vote.");

}

**Output:-**

You are eligible to vote.

**Ans(2):-**

**How switch statements work :-**

A **switch** statement is used to perform different actions based on different conditions. It compares a value against multiple cases and runs the code for the matching case. If no case matches, it runs the **default** block.

**Syntax:**

switch (expression) {

case value1:

// Code for value1

break;

case value2:

// Code for value2

break;

default:

// Code if no case matches

}

**Example:**

let day = "Monday";

switch (day) {

case "Monday":

console.log("Start of the work week.");

break;

case "Friday":

console.log("End of the work week.");

break;

default:

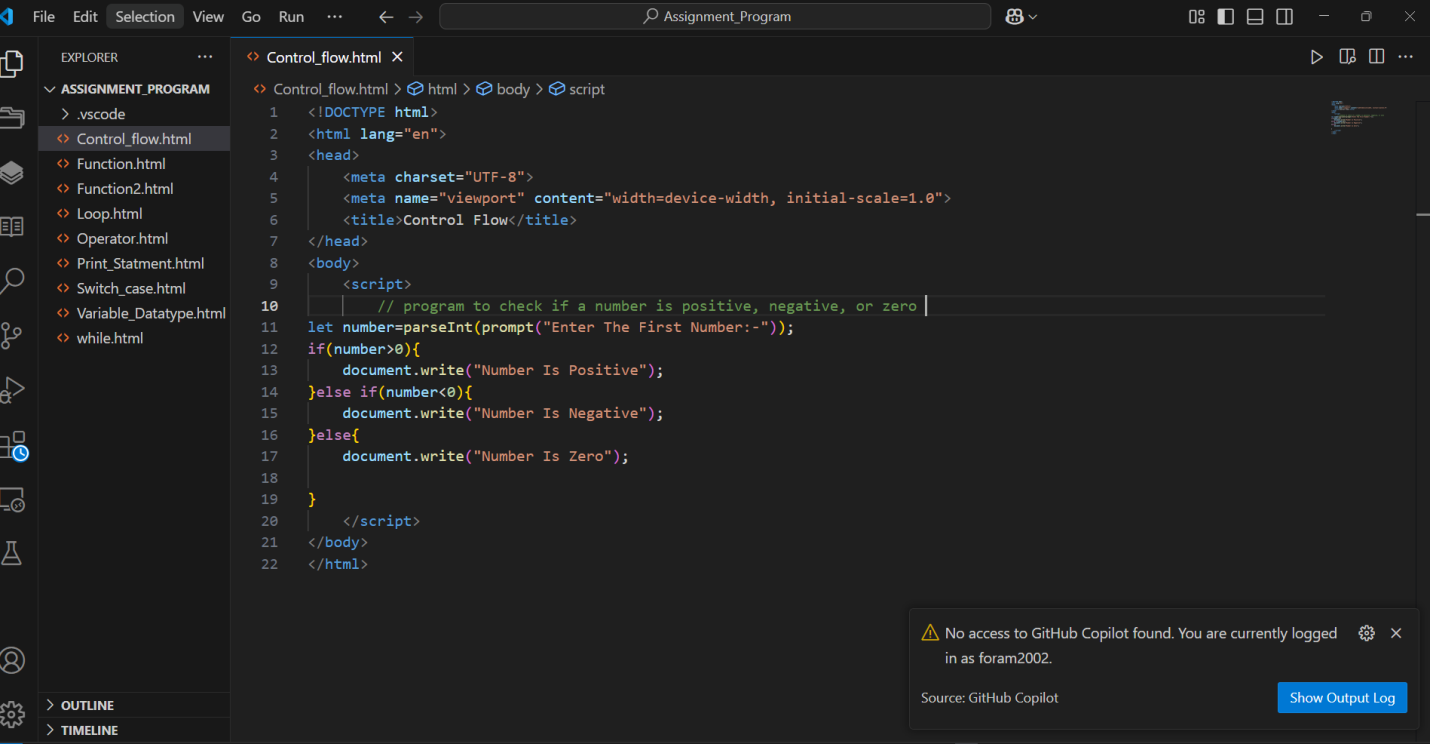
console.log("It's a regular day.");

}

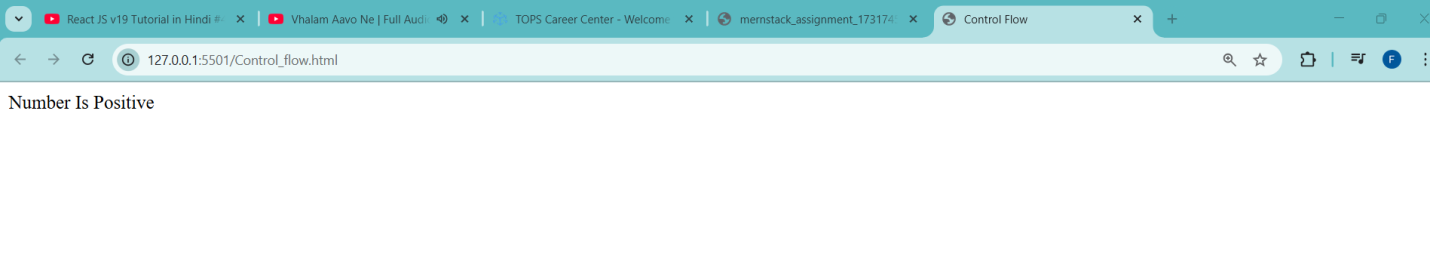
**Output:-**

Start of the work week

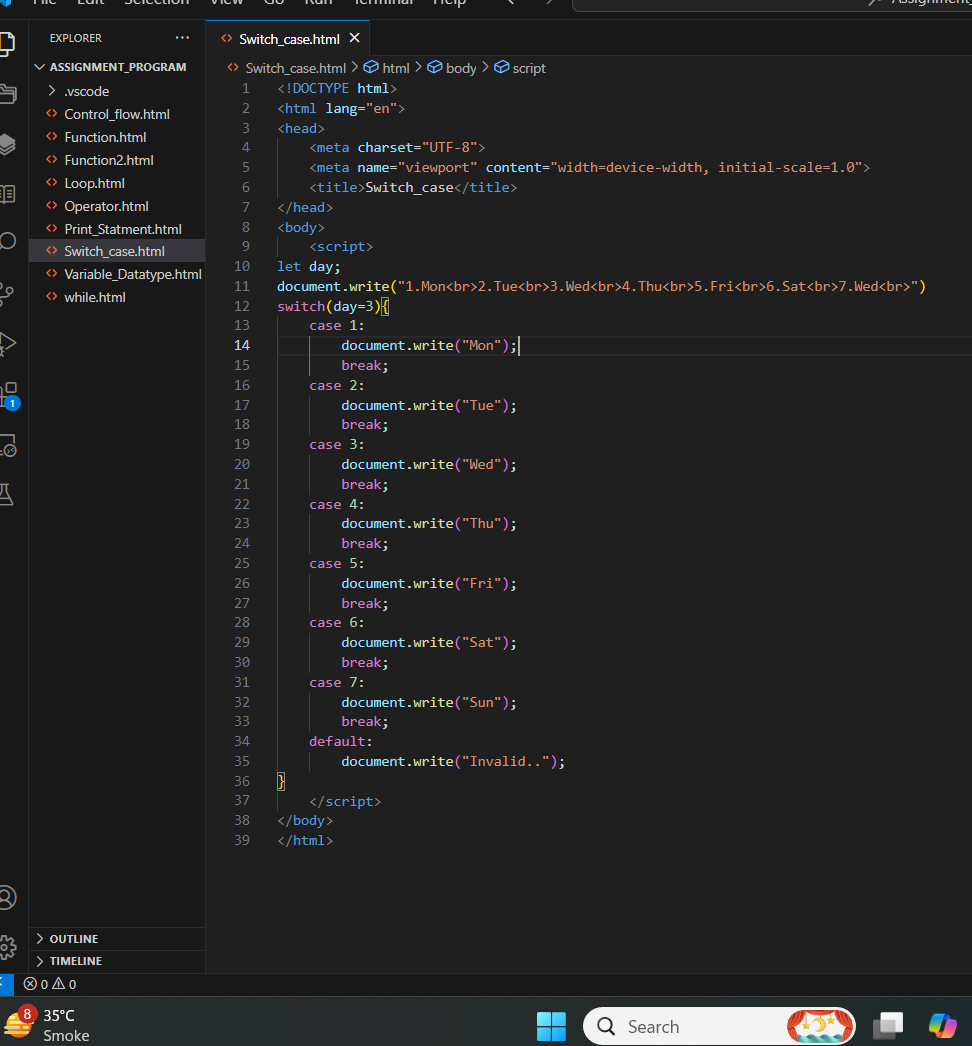
**Lab Assignment :-**

****

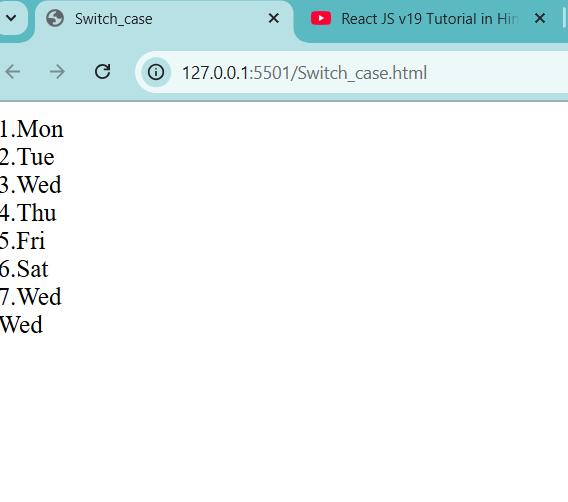
**Output:-**

****

**Task – 2:-**

****

**Output:-**

****

**Loops (For, While, Do-While) :-**

**Ans(1):-**

Loops are used to run a block of code multiple times. In JavaScript, there are three main types of loops: **for**, **while**, and **do-while.**

**1. For Loop:**

* A **for** loop runs a specific number of times, based on a condition.  
  **Structure:**

for (initialization; condition; increment/decrement) {

// Code to run

}

**Example:**

for (let i = 1; i<= 5; i++) {

console.log(i);

}

**Output:-**

1, 2, 3, 4, 5

**2. While Loop:**

* A **while** loop runs as long as a condition is **true**.

**Structure:**

while (condition) {

// Code to run

}

**Example:**

let i = 1;

while (i<= 5) {

console.log(i); i++;}

**Output:-**

1, 2, 3, 4, 5

**3. Do-While Loop**

* A **do-while** loop runs the code **at least once** before checking the condition.

**Structure:**

do {// Code to run

} while (condition);

**Example:**

let i = 1;

do {

console.log(i); i++;

} while (i<= 5);

**Output:-**

1, 2, 3, 4, 5

**Ans(2):-**

**Difference Between While and Do-While Loops:**

**While loop:-**

🡪 Checks the condition **before** the loop runs. May **not run at all** if the condition is false initially.

**Do-While loop:-**

->Checks the condition **after** the loop runs. Always runs the code **at least once**, even if the condition is false.

Example:-

let count = 0;

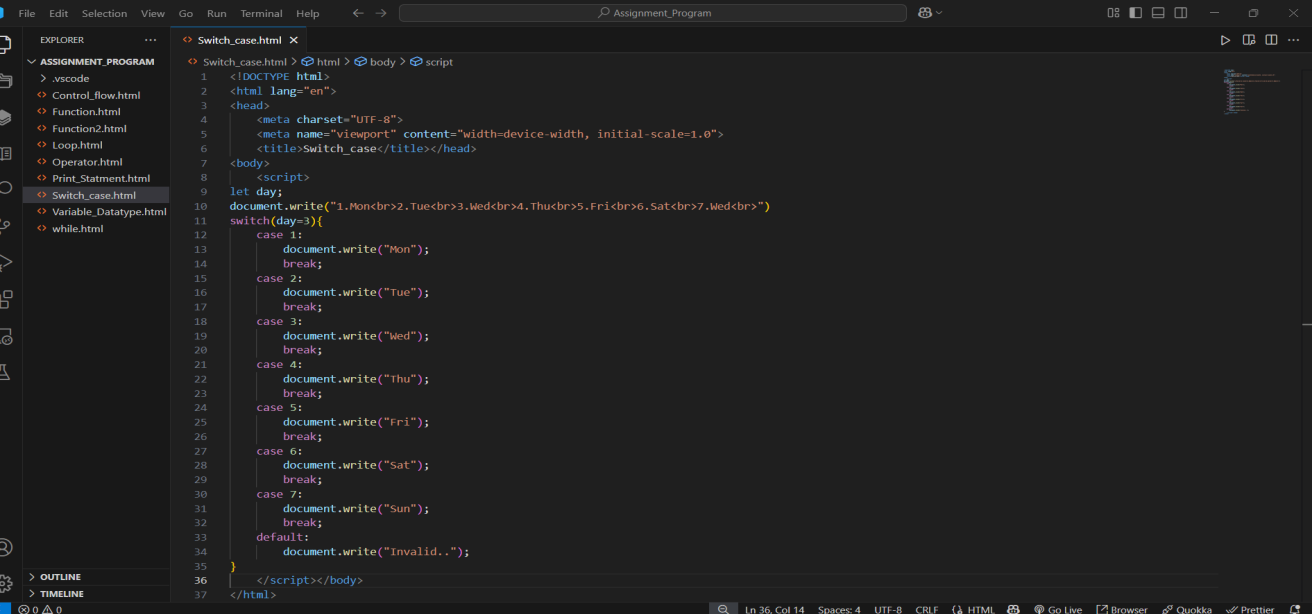
do {console.log(count);

} while (count > 0);

Output:-

Prints 0 once, even though the condition is false.

**Lab Assisnment:-**

****

**Output:-**

****

**Function:-**

🡪 Functions are **reusable blocks of code** designed to perform a specific task. They help make code modular, organized, and easy to maintain.

🡪 To declare a function, use the **function** keyword, followed by a name, parentheses, and a code block.

🡪 To call (or execute) a function, use its name followed by parentheses. Pass any required arguments inside the parentheses.

**Syntax:**

function functionName(parameters) {

// Code to run

}

**Example:**

function great(name) {

console.log("Hello, " + name + "!");

}

**Ans(2):-**

**Difference Between :-**

🡪A named function defined using the function keyword. function functionName() { ... }

🡪Function Expression-

|  |
| --- |
| A function assigned to a variable, can be anonymous. |
| 🡪 constfunctionName = function() { ... };  🡪 Can be called **before its declaration** due to hoisting.  🡪 Cannot be called before its definition.  **Examples:**  **Function Declaration:**  greet(); // Works due to hoisting  function greet() {  console.log("Hello!");  }  **Function Expression:**  greet(); // Error: greet is not defined yet  const greet = function() {  console.log("Hello!");  };  **Ans(3):-**   1. **Parameters:**  * Parameters are **placeholders** for values you pass into a function. * They allow the function to work with different inputs. * You define parameters in the parentheses when declaring a function.   **Example:**  function add(a, b) { // 'a' and 'b' are parameters  return a + b;  }  console.log(add(3, 5)); // Output: 8   1. **Return Values:**  * A **return value** is the result a function sends back after it finishes running. * You use the return keyword to specify the value to be returned. * If there’s no return, the function returns undefined by default.   **Example:**  function multiply(x, y) {  return x \* y; // Returns the product of x and y  }  let result = multiply(4, 6); // result = 24  console.log(result); // Output: 24  **Explanation:**  **Parameters**: Input values for the function (e.g., a, b in add(a, b)).  **Return Values**: Output of the function sent back to the caller using return.  **Lab Assignment:-**    **Output:-**    **Task – 2:-**    **Output:-**    **Array:-**  **Ans(1):-**  An **array** is a special data structure in JavaScript used to store multiple values in a single variable. The values (called elements) are stored in an ordered way, and each element has an index starting from 0.  **Declaring and Initializing an Array:**   1. **Declare an Array:** Use square brackets [] to create an array.   **Syntax:**  let arrayName = [];   1. **Initialize an Array with Values:** You can add elements to the array when you declare it.   **Syntax:**  let arrayName = [value1, value2, value3];  **Examples:**  **Empty Array:**  let fruits = [];  **Array with Values:**  let fruits = ["Apple", "Banana", "Cherry"];  console.log(fruits); // Output: ["Apple", "Banana", "Cherry"]  **Accessing Elements:**  console.log(fruits[0]); // Output: "Apple" (first element)  console.log(fruits[1]); // Output: "Banana"  **Ans(2):-** **Push()**  * Adds one or more elements **to the end** of the array. * The new length of the array. * **Example:**   let fruits = ["Apple", "Banana"];  fruits.push("Cherry");  console.log(fruits); // Output: ["Apple", "Banana", "Cherry"] **Pop()**  * Removes the **last element** from the array. * The removed element. * **Example:**   let fruits = ["Apple", "Banana", "Cherry"];  let removed = fruits.pop();  console.log(fruits); // Output: ["Apple", "Banana"]  console.log(removed); // Output: "Cherry" **Shift()**  * Removes the **first element** from the array. * The removed element. * **Example:**   let fruits = ["Apple", "Banana", "Cherry"];  let removed = fruits.shift();  console.log(fruits); // Output: ["Banana", "Cherry"]  console.log(removed); // Output: "Apple" **Unshift()**  * Adds one or more elements **to the beginning** of the array. * The new length of the array.   **Example:**  let fruits = ["Banana", "Cherry"];  fruits.unshift("Apple");  console.log(fruits); // Output: ["Apple", "Banana", "Cherry"]  **Lab Assignment:-**    **Output:-**    **Task-(2)**  **:-**  **Output:-**    **Objects:-**  **Ans(1):-**  An **object** in JavaScript is a data structure that stores data in the form of **key-value pairs**. Objects are used to represent real-world entities with properties and behaviors.  **Syntax:**  **let objectName = {**  **key1: value1, key2: value2,**  **};**  **Example:**  let person = {  name: "Alice",  age: 25,  greet: function () {  console.log("Hello!");  },  };  console.log(person.name); // Output: Alice  person.greet(); // Output: Hello!  **Ans(2):-**  🡪 You can access and update object properties using **dot notation** or **bracket notation.**  **🡪** When the property name is a simple string (e.g., no spaces or special characters).  Example:-  let person = { "first name": "Alice", age: 25 };  console.log(person["first name"]); // Output: Alice  console.log(person["age"]); // Output: 25  **Lab Assignment:-**    **Output:-**    **JavaScript Event:-**  **Ans(1):-**  **Events** in JavaScript are actions or occurrences that happen in the browser, like:   * A user clicking a button. * Typing in an input field. * Hovering over an element.   JavaScript can respond to these events to make web pages interactive.  An **event listener** is a way to tell the browser to watch for a specific event on an element and run a function when that event occurs.  **Role of Event Listeners:**   * They connect actions (like clicking) with the code you want to run. * Allow dynamic and responsive behavior without modifying the HTML.   **Ans(2):-**  The addEventListener() method in JavaScript is used to attach an event to an element. When the event occurs, a specified function is executed. **Syntax:** **JAVASCRIPT:**  element.addEventListener(eventType, function);  🡪**element**: The HTML element (e.g., button).  🡪**eventType**: The type of event (e.g., "click").  **🡪function**: The function to execute when the event occurs. **Example:**  * **HTML:**   <button id="myButton">Click Me</button>   * **JAVASCRIPT:**   let button = document.getElementById("myButton");  button.addEventListener("click", function() {  console.log("Button clicked!");  });  **Lab Assignment**  **Output:-**    **DOM Manipulation:-**  **Ans(1):-**  The **DOM** is a programming interface for web documents. It represents the HTML structure of a webpage as a tree of objects, where each element is an object that can be manipulated.  **Interact with the DOM:**JavaScript can **access**, **modify**, **add**, or **remove** HTML elements and their content using the DOM. This allows dynamic changes to the webpage without reloading it.  **Example:**  <p id="demo">Hello, World!</p>  <button onclick="changeText()">Click Me</button>  <script>  function changeText() {  document.getElementById("demo").innerText = "Text changed!";}  </script>  **Ans(2):-**  **Methods to Select DOM Elements in JavaScript:**   1. **getElementById():**  * Selects an element by its **ID**. * Returns **one element**. * **Example:**   let element = document.getElementById("myId");   1. **getElementsByClassName():**  * Selects elements by their **class name**. * Returns a **live HTMLCollection** of elements. * **Example:**let elements = document.getElementsByClassName("myClass");  1. **querySelector():**  * Selects the **first element** that matches the CSS selector (ID, class, tag, etc.). * Returns **one element**. * **Example:**let element = document.querySelector(".myClass"); // or #myId, p, etc.   **Lab Assignment:-**    **Output:-**     1. **setTimeout():-**Executes a function **once** after a specified delay (in milliseconds). It is used to **delay** the execution of a function for a specific period.  * **Example**: If you want to show a message after 3 seconds, you can use setTimeout():   + - setTimeout(() => {       * console.log("This runs after 3 seconds");     - }, 3000);  1. **setInterval():-**Executes a function **repeatedly** at a specified interval (in milliseconds).It is used to **repeat** the execution of a function at regular intervals.  * **Example**: If you want to log a message every 2 seconds, you can use setInterval():   + - setInterval(() => {       * console.log("This runs every 2 seconds");     - }, 2000);   **Ans(2):-**  **Example of setTimeout() to Delay an Action by 2 Seconds:**  setTimeout(function() {  console.log("This runs after 2 seconds");  }, 2000);  **JavaScript Error Handling:-**  **Ans(1):-**  🡪 Error handling in JavaScript helps manage and control errors that may occur in a program. Instead of stopping the program completely, error handling allows us to "catch" the error and handle it properly.  🡪**try Block** → Code that may cause an error is written inside try.  🡪**catch Block** → If an error occurs in try, catch handles the error.  🡪**finally Block** (optional) → Always runs, whether there was an error or not.  Example:-  try { let num = 10;  console.log(num.toUpperCase()); // This will cause an error  } catch (error) {  console.log("An error occurred:", error.message);  } finally {  console.log("This runs no matter what.");  }  **Ans(2):-**  **🡪** Error handling is important because it helps keep JavaScript application,**running smoothly** even when something goes wrong.  🡪 If an error occurs, the whole app **doesn’t stop working**—we can handle the error and continue.  🡪 Catching errors helps developers **find and fix bugs** faster. If a website tries to **fetch data from an API** but the internet is slow, error handling ensures the page **doesn’t break**.  **Lab Assignment:-**    **Output:-** |
|  |