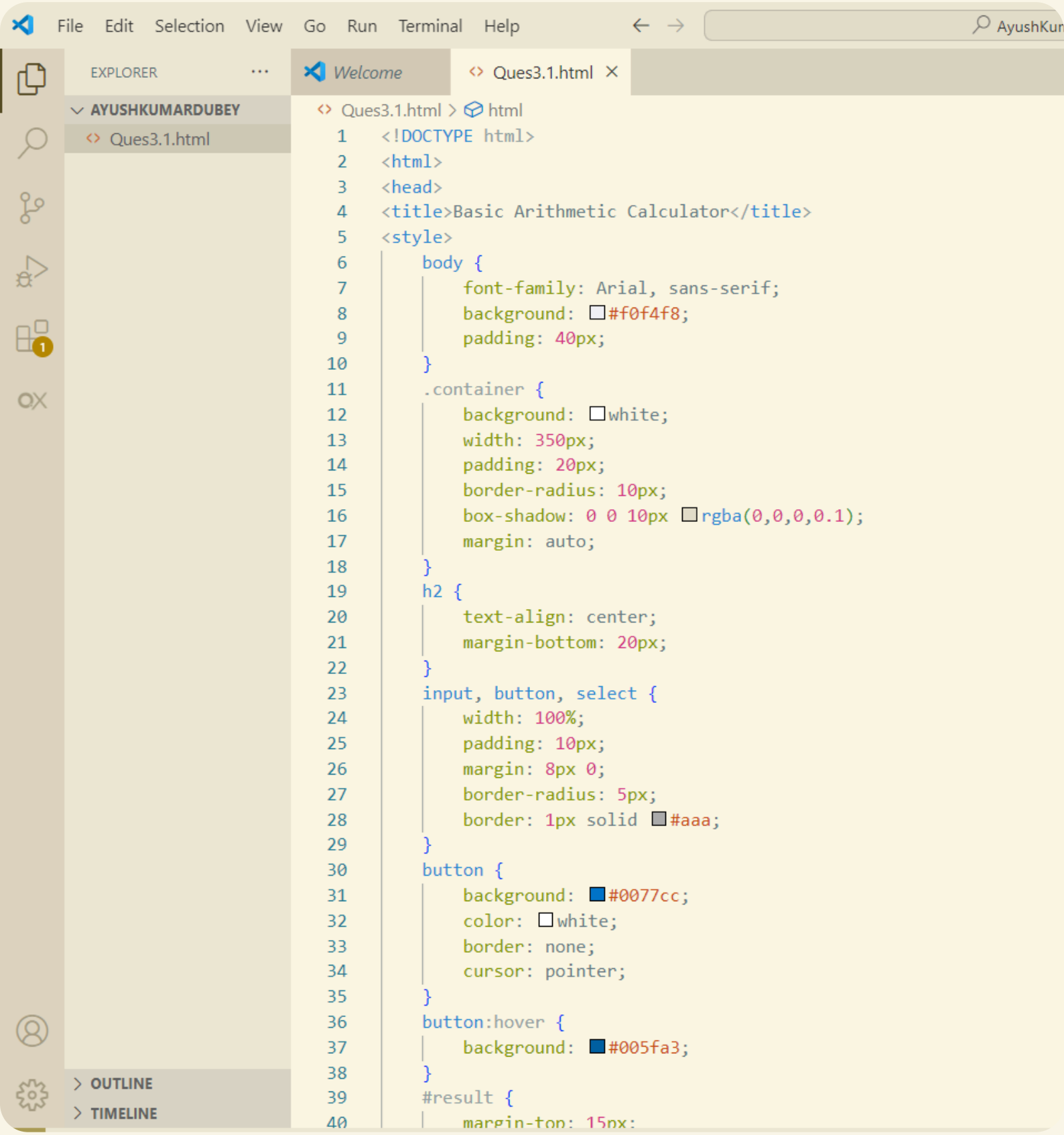
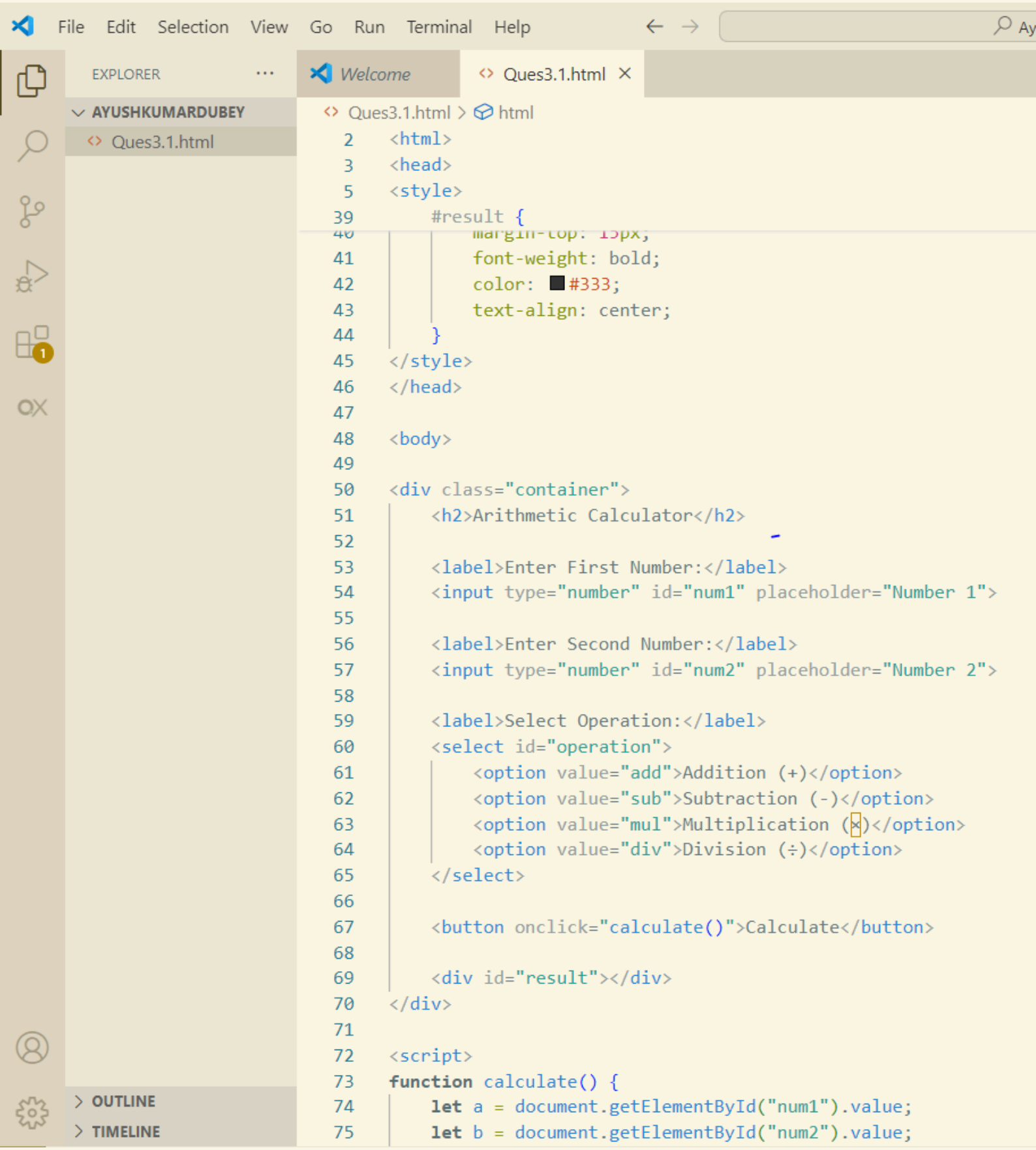
3.1 Create a JavaScript-enabled web page that performs basic arithmetic operations using input from users

Introduction

A JavaScript-based arithmetic page allows users to enter two numbers and choose an operation such as addition, subtraction, multiplication, or division. The browser performs calculations instantly without the need for external processing. This makes the page interactive and helps students understand how JavaScript interacts with HTML elements.




```
<!DOCTYPE html>
<html>
<head>
<title>Basic Arithmetic Calculator</title>
<style>
    body {
        font-family: Arial, sans-serif;
        background: #f0f4f8;
        padding: 40px;
    }
    .container {
        background: white;
        width: 350px;
        padding: 20px;
        border-radius: 10px;
        box-shadow: 0 0 10px rgba(0,0,0,0.1);
        margin: auto;
    }
    h2 {
        text-align: center;
        margin-bottom: 20px;
    }
    input, button, select {
        width: 100%;
        padding: 10px;
        margin: 8px 0;
        border-radius: 5px;
        border: 1px solid #aaa;
    }
    button {
        background: #0077cc;
        color: white;
        border: none;
        cursor: pointer;
    }
    button:hover {
        background: #005fa3;
    }
    #result {
        margin-top: 15px;
```



Input Explanation → The webpage accepts:

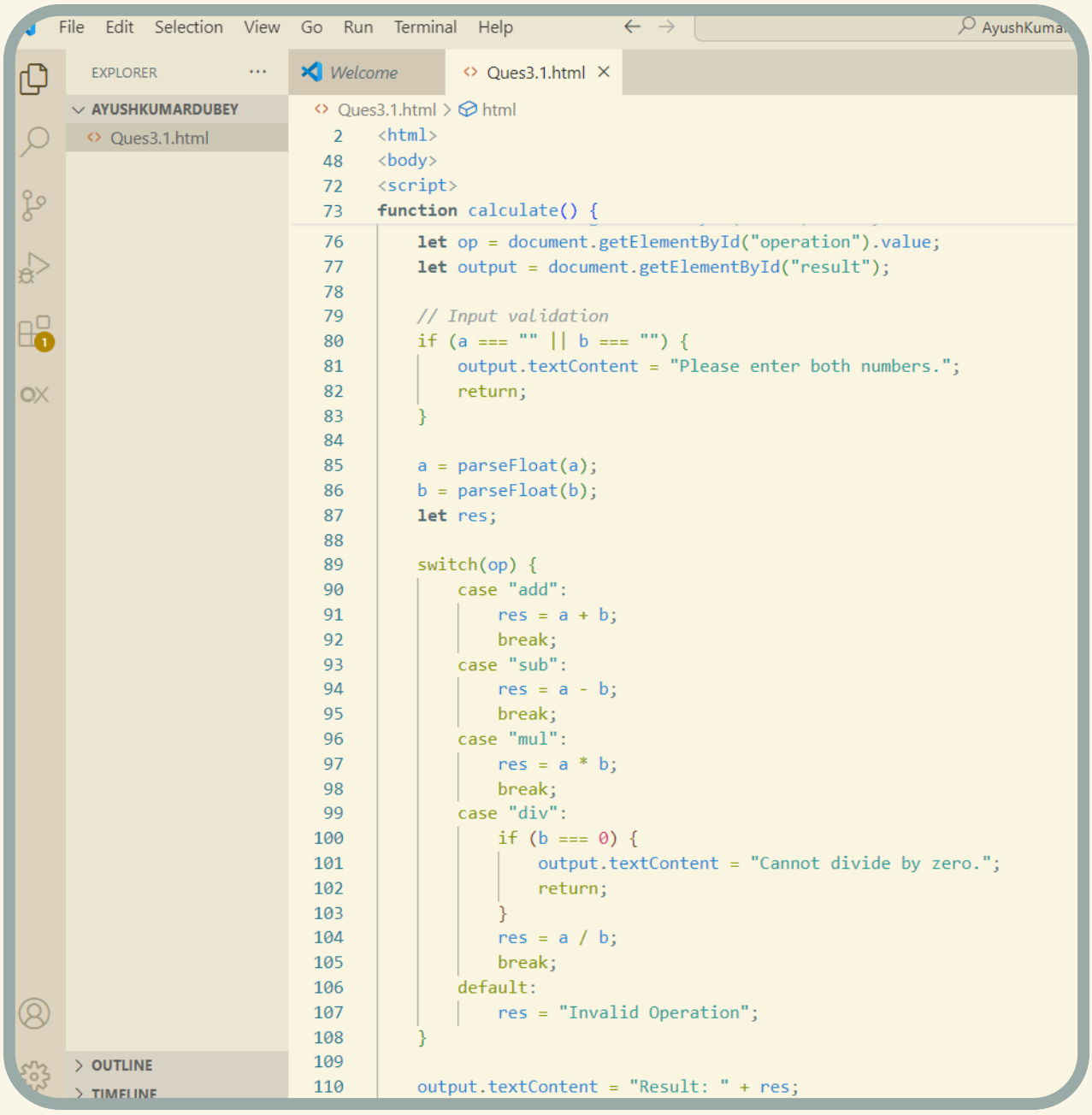
Input Purpose

Number 1: First value to be calculated

Number 2: Second value to be calculated

Operation Button: Chooses the type of arithmetic

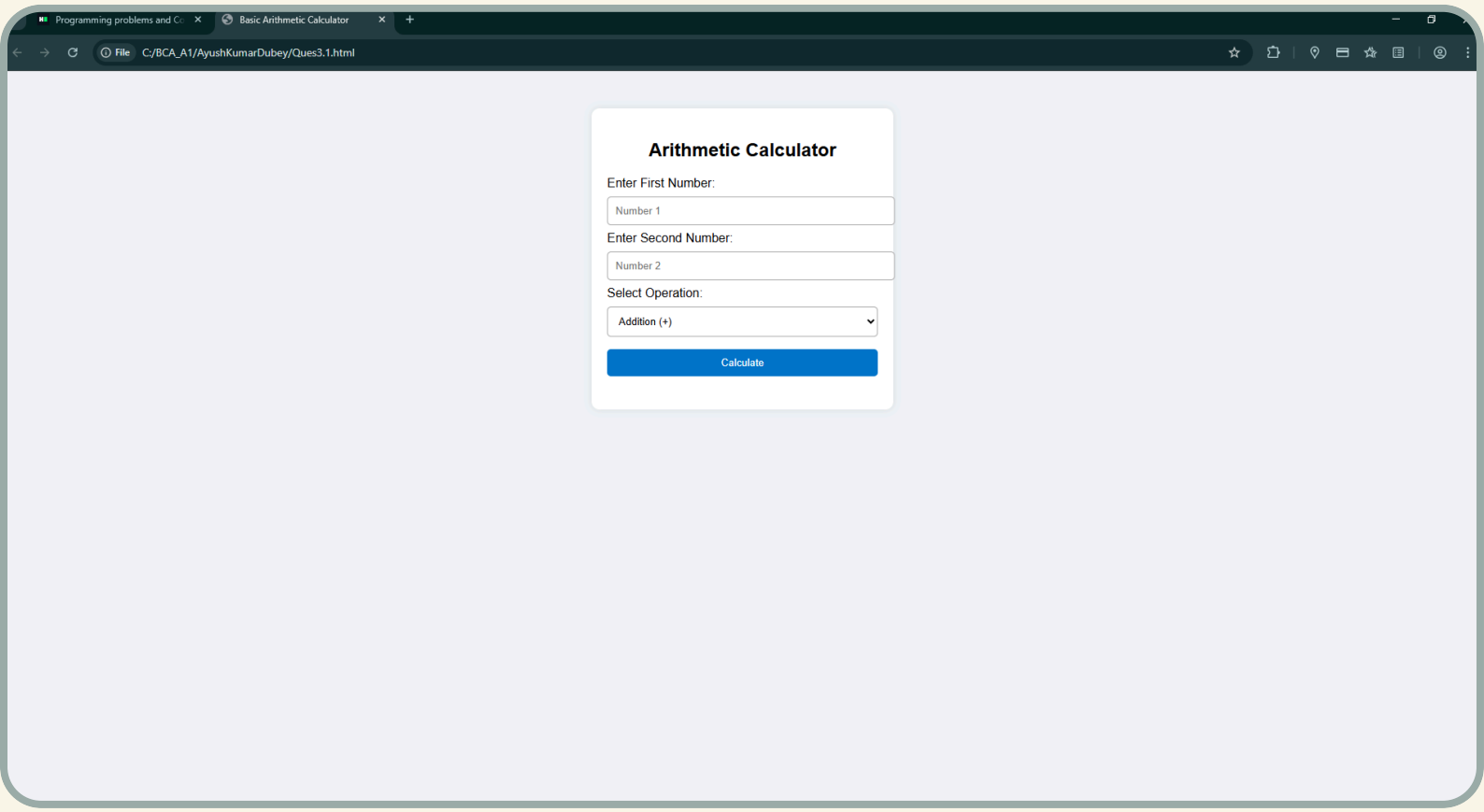
JAVASCRIPT HERE ⬇️

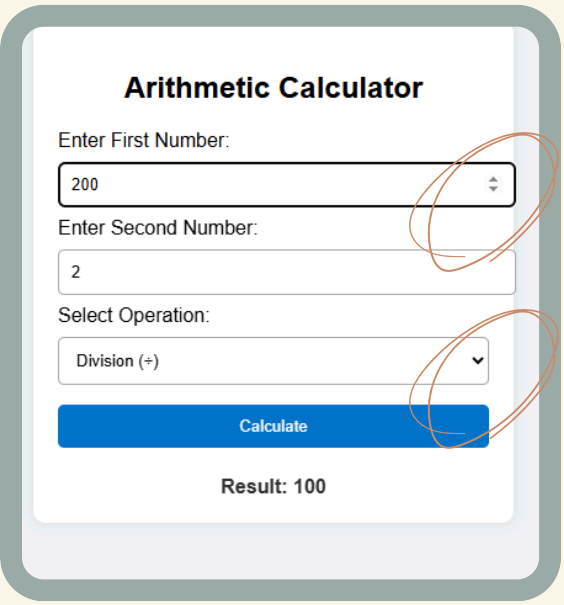


```
<html>
<body>
<script>
function calculate() {
    let op = document.getElementById("operation").value;
    let output = document.getElementById("result");

    // Input validation
    if (a === "" || b === "") {
        output.textContent = "Please enter both numbers.";
        return;
    }

    a = parseFloat(a);
    b = parseFloat(b);
    let res;

    switch(op) {
        case "add":
            res = a + b;
            break;
        case "sub":
            res = a - b;
            break;
        case "mul":
            res = a * b;
            break;
        case "div":
            if (b === 0) {
                output.textContent = "Cannot divide by zero.";
                return;
            }
            res = a / b;
            break;
        default:
            res = "Invalid Operation";
    }

    output.textContent = "Result: " + res;
```

```
            res = "Invalid Operation";
        }

        output.textContent = "Result: " + res;
    }
    </script>

    </body>
    </html>
```

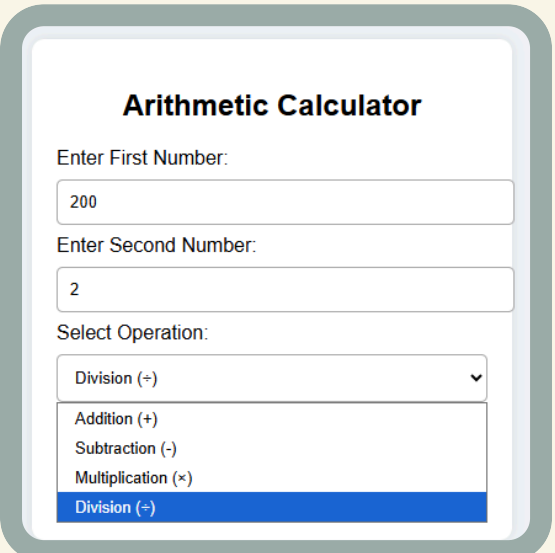
OUTPUT





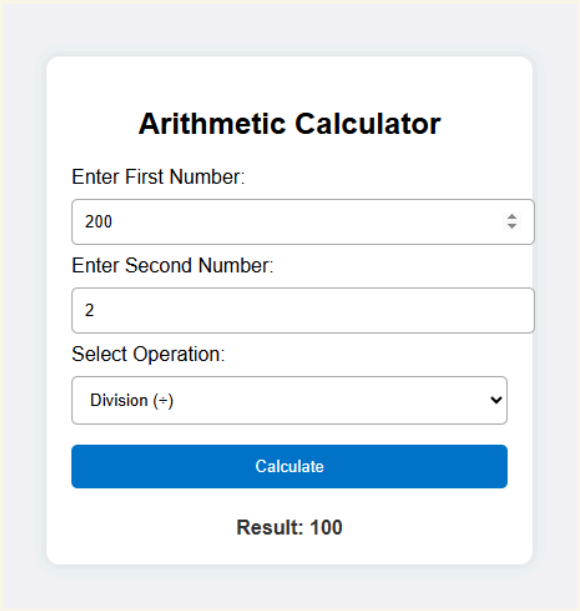
| To Increase or Decrease Value

| To Switch operation Ex:- (-) to (+)

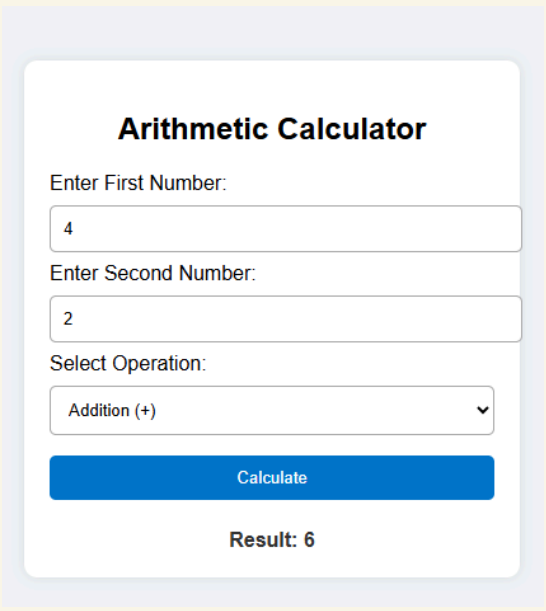


| Types Of operations which are available use as accordingly ;

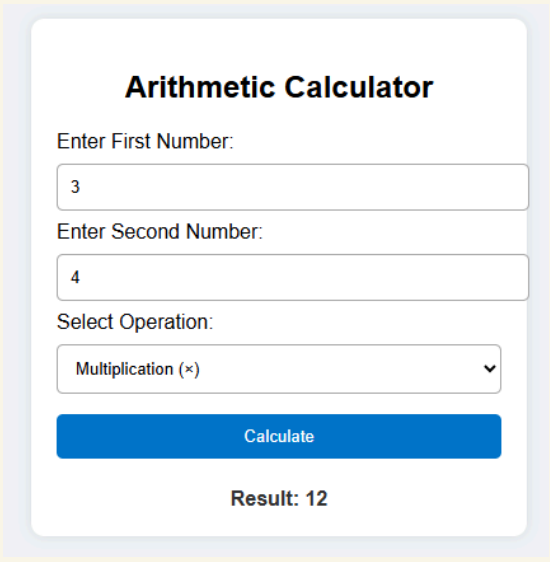
OUTPUT



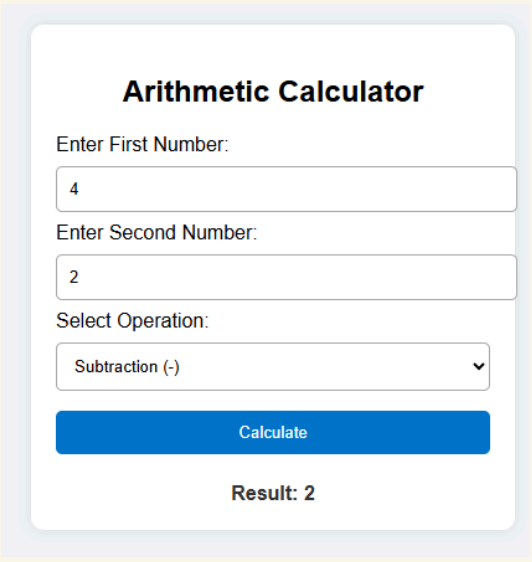
It showcasing division



Here Example of Addition



Here Ex. For Multiplication



Subtracting two numbers

OUTPUT

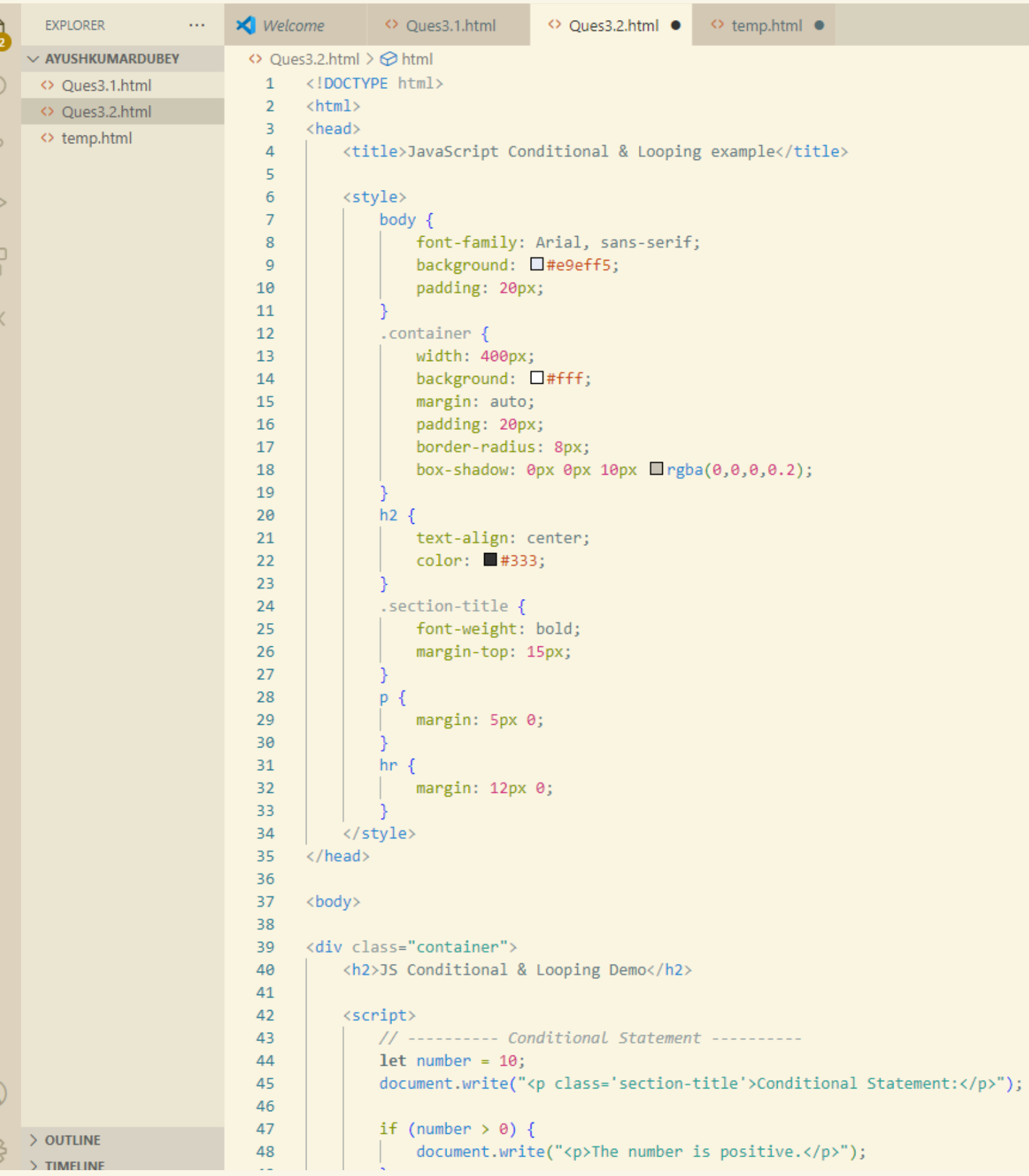
Data Types Used

Data Type	Role
number	Stores numeric user inputs
string.	Used for identifying the selected operation
HTMLElement.	Interacts with input/output elements

Conclusion

This example demonstrates how to combine HTML and JavaScript to build a simple interactive calculator. It teaches key concepts such as DOM access, input handling, event-driven functions, and basic arithmetic logic. Students can extend the project by adding CSS, dropdown menus, or automated validation.

Q3.2 Write JavaScript to demonstrate conditional statements and looping (for, while, do...while).



Conditional statements help a program make decisions based on conditions. Loops allow repeated execution of code until a stopping point is reached. Together, they enable logical flow and controlled repetition in JavaScript.

JAVASCRIPT HERE

Ques3.1.html
Ques3.2.html
temp.html

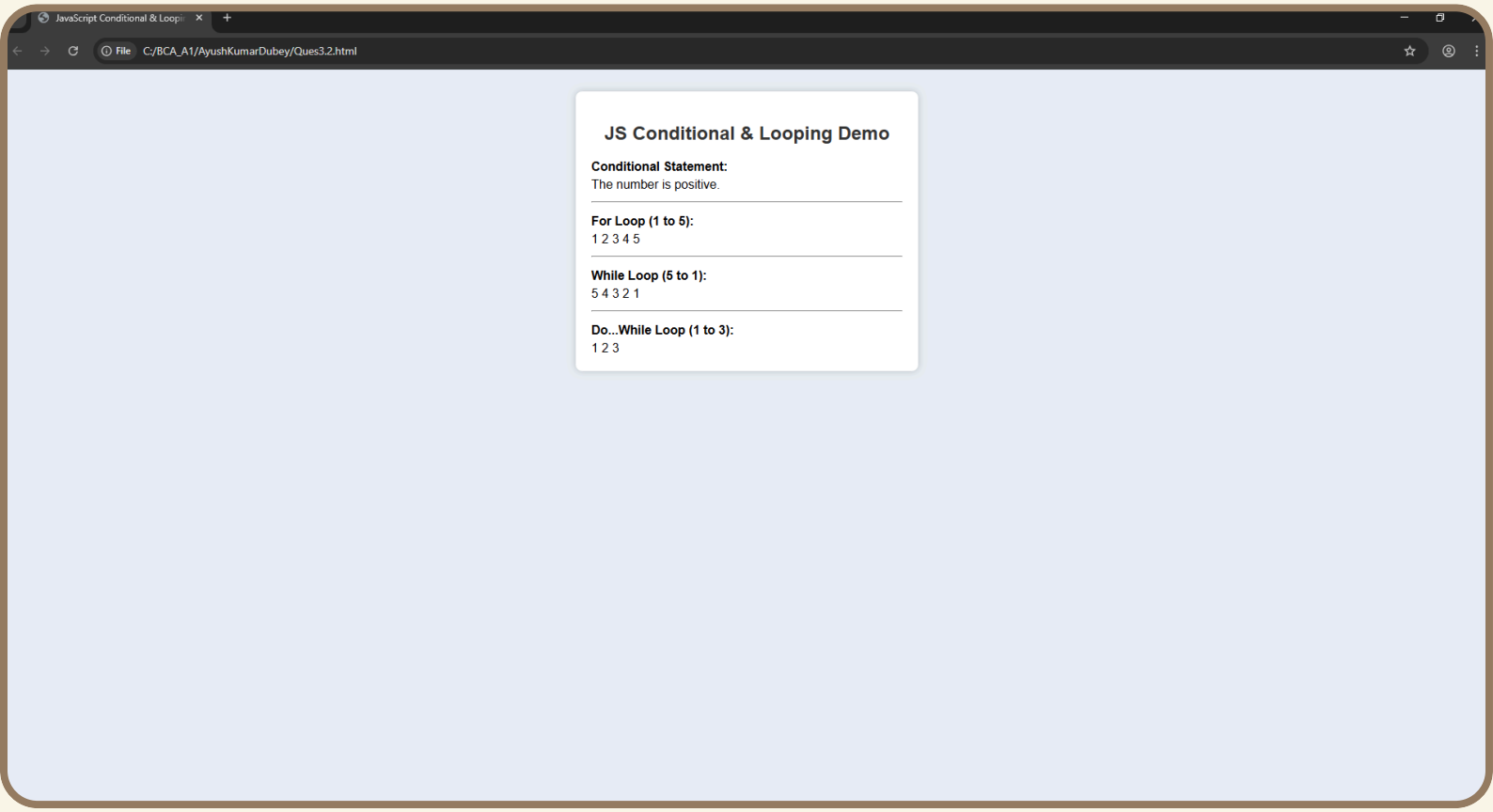
```
<html>

<body>

<div class="container">
    <h2>JS Conditional & Looping Demo</h2>

    <script>
        // ---------- Conditional Statement ----------
        let number = 10;
        document.write("<p class='section-title'>Conditional Statement:</p>");

        if (number > 0) {
            document.write("<p>The number is positive.</p>");
        }
        else if (number < 0) {
            document.write("<p>The number is negative.</p>");
        }
        else {
            document.write("<p>The number is zero.</p>");
        }

        document.write("<hr>");

        // ---------- For Loop ----------
        document.write("<p class='section-title'>For Loop (1 to 5):</p>");
        for (let i = 1; i <= 5; i++) {
            document.write(i + " ");
        }

        document.write("<hr>");
        document.write("<p class='section-title'>While Loop (5 to 1):</p>");
        let x = 5;
        while (x >= 1) {
            document.write(x + " ");
            x--;
        }
        document.write("<hr>");
        document.write("<p class='section-title'>Do...While Loop (1 to 3):</p>
        let y = 1;
        do {
            document.write(y + " ");
            y++;
        } while (y <= 3);
    </script>
</div>
</body>
</html>
```

OUTLINE

How it works

The program checks each condition. The first true condition produces the output. Only one block executes.

OUTPUT



A. For Loop

```
 For (let i = 1; i <= 5; i++) {
   console.log("For loop count: " + i);
}
```

Explanation

Starts at 1, stops at 5

Increments by 1

Prints 5 lines

B. While Loop

```
let count = 1;
while (count <= 5) {
   console.log("While loop count: " + count);
   count++;
}
```

Explanation

Checks the condition before each repeated action

Stops when the condition becomes false

C. Do...While Loop

```
let num = 1;
do {
   console.log("Do...While loop count: " + num);
   num++;
} while (num <= 5);
```

Explanation

Executes the block at least once, even if the condition is false initially

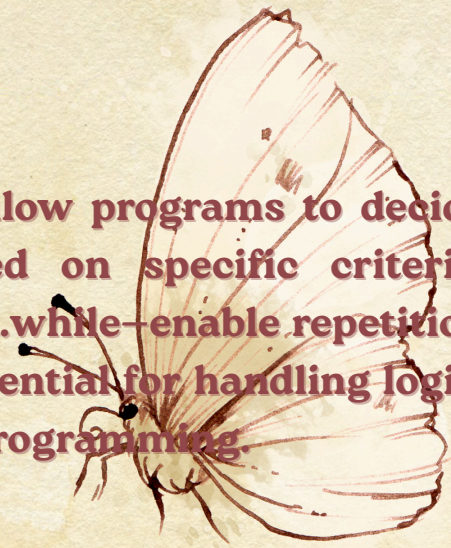
OUTPUT

| 4. Summary of Data Types Used

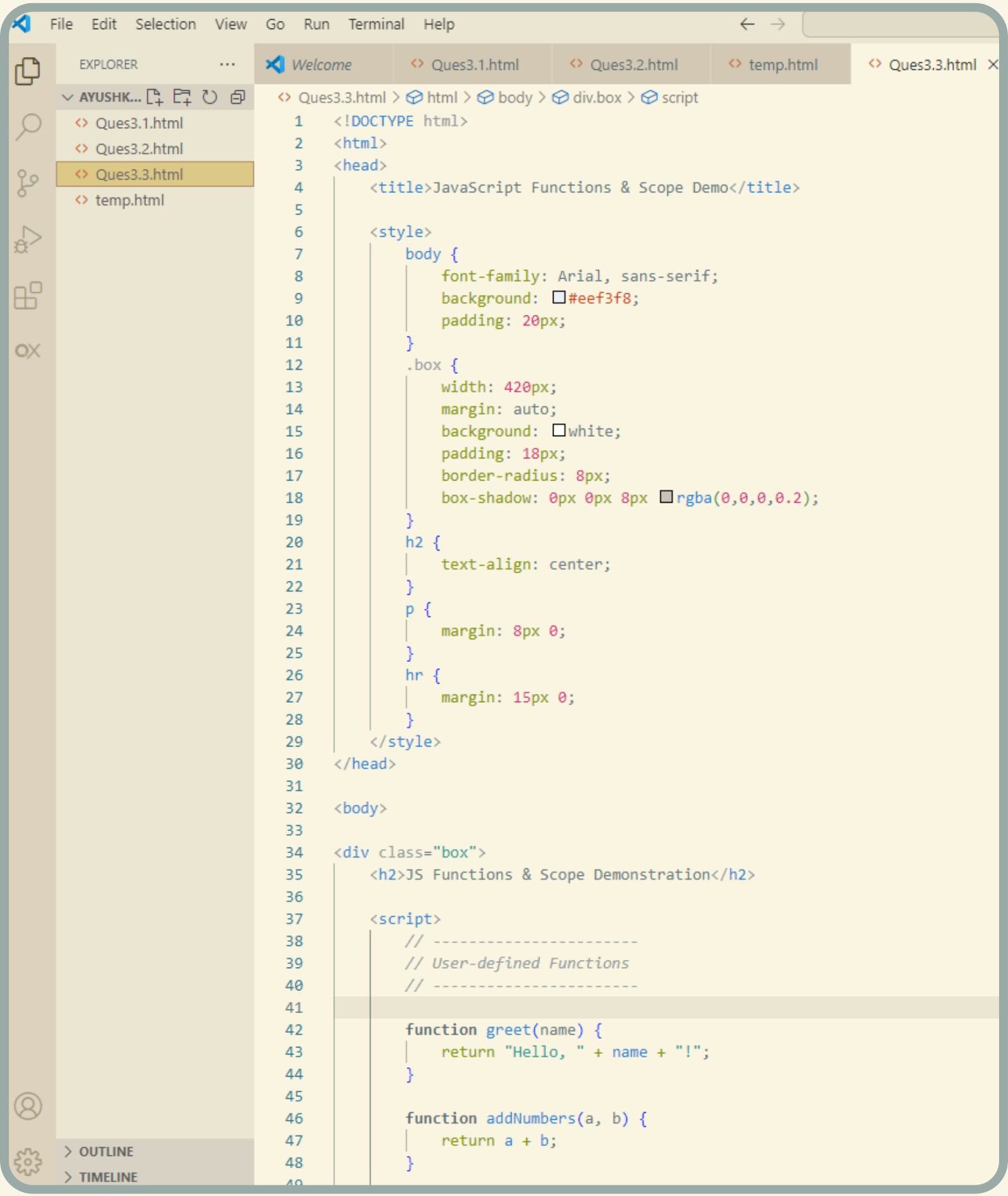
Data Type	Used For
number	Loop counters, score values
string	Console messages
boolean	Internal conditional checking outcome

Conclusion

JavaScript's conditional statements allow programs to decide which instructions to execute based on specific criteria. Looping structures—for, while, and do...while—enable repetition of tasks efficiently. These tools are essential for handling logic, automation, and dynamic behavior in programming.



3.3 Create and invoke user-defined functions in JavaScript use var, let, and const for scope demonstration

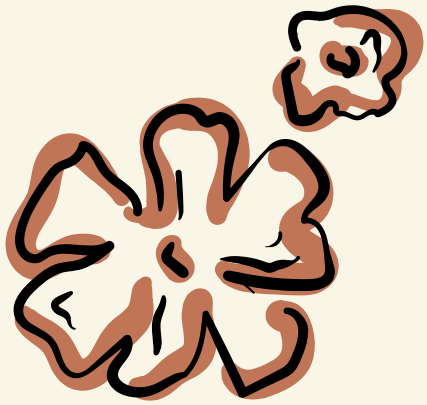


User-defined functions allow programmers to bundle instructions into reusable blocks. JavaScript also provides different ways to declare variables—var, let, and const—each with its own scope rules. Understanding these helps prevent errors and improves code organization.

Creating and Invoking User-Defined Functions

A. Simple Function Example

```
function greet(name) {
  return "Hello, " + name + "!";
}
console.log(greet("James")); // Function call
```



Explanation

*** Defined using the function keyword**

*** Accepts a parameter**

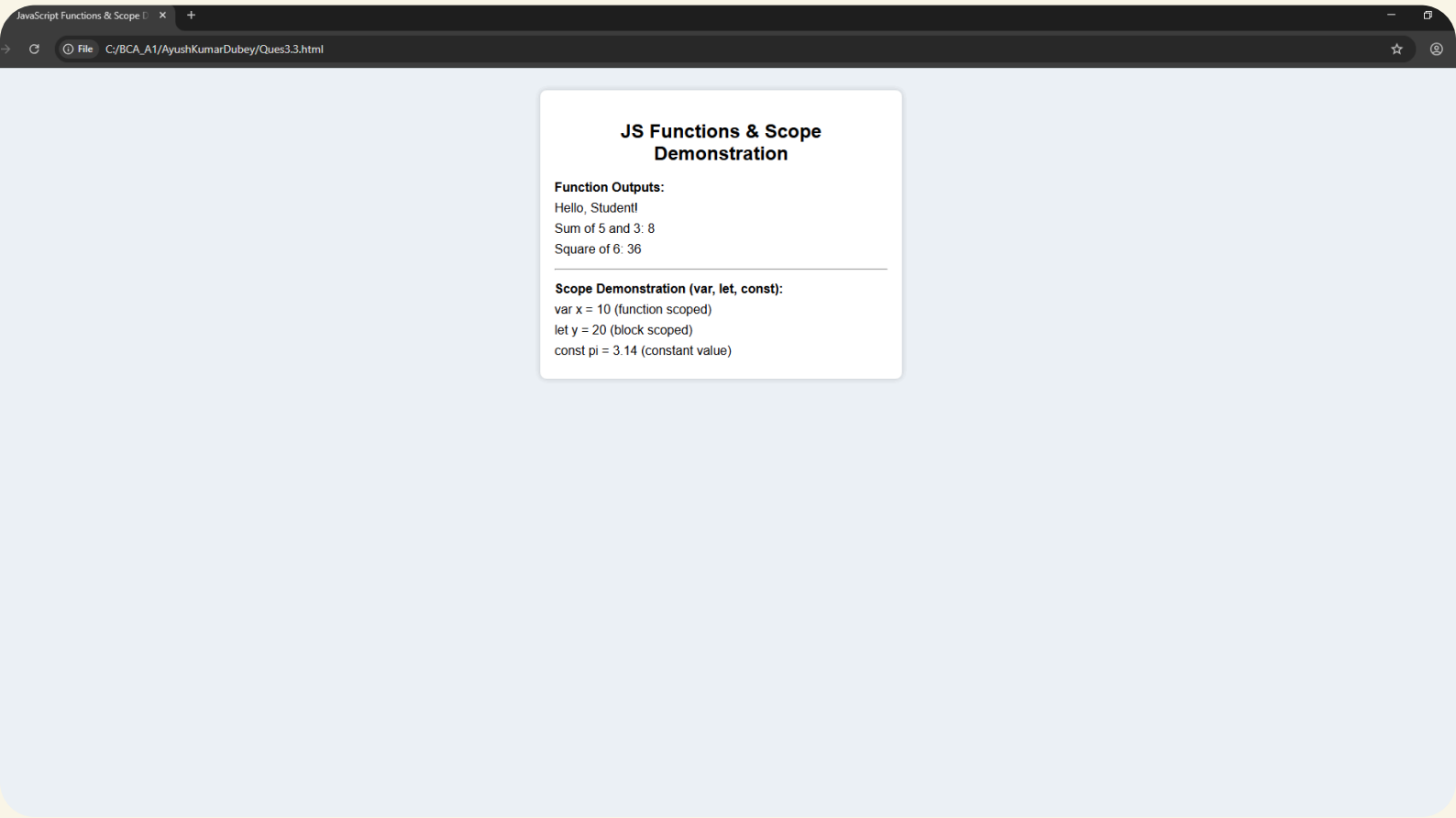
*** Returns a value**

*** Invoked using its name and passing an argument**

```
49
50          function showSquare(n) {
51              return n * n;
52          }
53
54          document.write("<p><b>Function Outputs:</b></p>");
55          document.write("<p>" + greet("Student") + "</p>");
56          document.write("<p>Sum of 5 and 3: " + addNumbers(5, 3) + "</p>");
57          document.write("<p>Square of 6: " + showSquare(6) + "</p>");
58
59          document.write("<hr>");
60
61          // ----------------------
62          // var, let, const example
63          // ----------------------
64
65          document.write("<p><b>Scope Demonstration (var, let, const):</b></p>"
66
67          // var example (function-scoped)
68          var x = 10;
69          document.write("<p>var x = " + x + " (function scoped)</p>");
70
71          // let example (block-scoped)
72          {
73              let y = 20;
74              document.write("<p>let y = " + y + " (block scoped)</p>");
75          }
76          // y is not accessible here (scope ends)
77
78          // const example (cannot be reassigned)
79          const pi = 3.14;
80          document.write("<p>const pi = " + pi + " (constant value)</p>");
81
82          // Trying to reassign const pi would cause an error
83      </script>
84
85  </div>
86
87  </body>
88  </html>
```

> OUTLINE
> TIMELINE

OUTPUT



Using var, let, and const for Scope Demonstration

Keyword	Scope	Can Reassign?	Can Redeclare
var	Function	Yes	No
let	Block	Yes	No
const	Block	No	No

Conclusion

User-defined functions are essential for organizing reusable logic in JavaScript. Variable declarations—var, let, and const—play a crucial role in controlling scope and preventing unintended value changes. Understanding function creation, function invocation, and scope helps developers write clean, predictable, and professional-grade JavaScript code.

Q. 3.4 Handle HTML form validation using JavaScript events like onsubmit, on input, and use alert() for feedback

Form validation ensures that users enter correct and complete information before the form is submitted. JavaScript provides several events to manage validation, including:

oninput → Runs as the user types

onsubmit → Runs when the user tries to submit the form

alert() → Displays messages and warnings

These events help prevent invalid data and give real-time feedback.

```
48
49  <body>
50
51  <div class="form-box">
52      <h2>Form Validation Demo</h2>
53
54      <form id="myForm" onsubmit="return validateForm()">
55
56          <label>Name:</label>
57          <input type="text" id="name" oninput="checkName()">
58          <small id="nameError">Name must be at least 3 characters</small>
59
60          <label>Email:</label>
61          <input type="text" id="email" oninput="checkEmail()">
62          <small id="emailError">Enter a valid email</small>
63
64          <label>Age:</label>
65          <input type="number" id="age" oninput="checkAge()">
66          <small id="ageError">Age must be 18 or above</small>
67
68          <button type="submit">Submit</button>
69      </form>
70  </div>
71
72  <script>
73      // -------- oninput validation --------
74
75      function checkName() {
76          let name = document.getElementById("name").value;
77          let err = document.getElementById("nameError");
78
79          if (name.length < 3) {
80              err.style.display = "block";
81              return false;
82          } else {
83              err.style.display = "none";
84              return true;
85          }
86      }
87
88      function checkEmail() {
89          let email = document.getElementById("email").value;
90          let err = document.getElementById("emailError");
91
92          if (!email.includes("@") || !email.includes(".")) {
93              err.style.display = "block";
```

3. Explanation of Events Used

A. oninput Event

Runs every time the user types.
Used for live validation.

Example:

oninput="checkName()"

B. onsubmit Event

Executes when the form is submitted.

Used for final form checking.

Example:

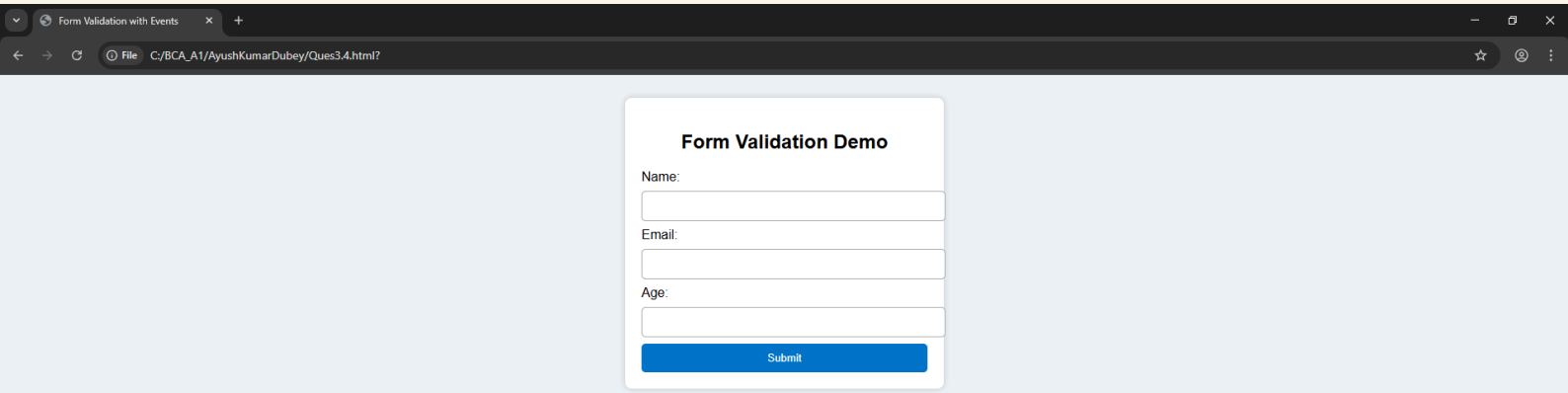
<form onsubmit="return validateForm()">

C. alert() Function

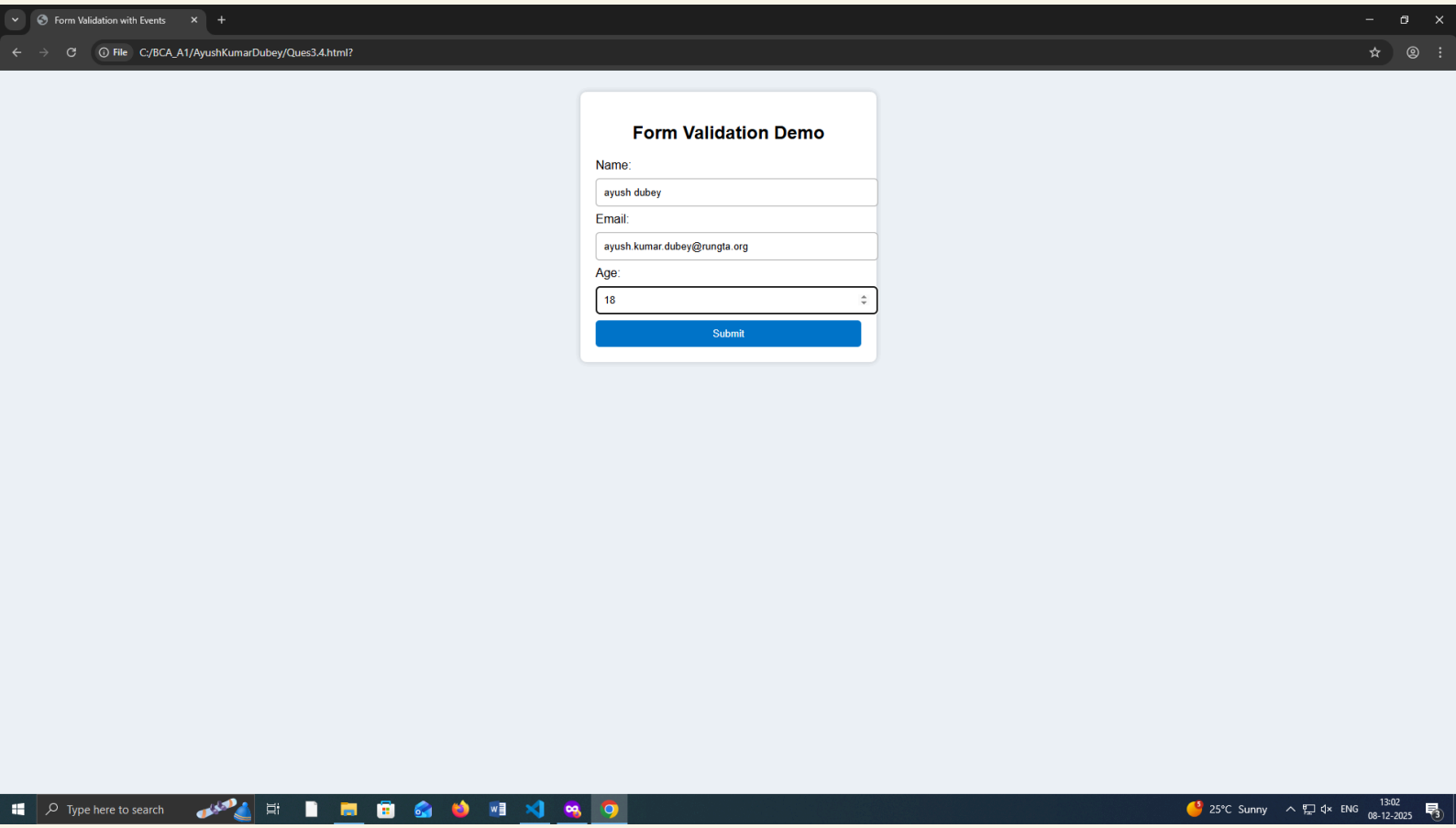
Shows pop-up messages.

Used to notify users about errors or success.

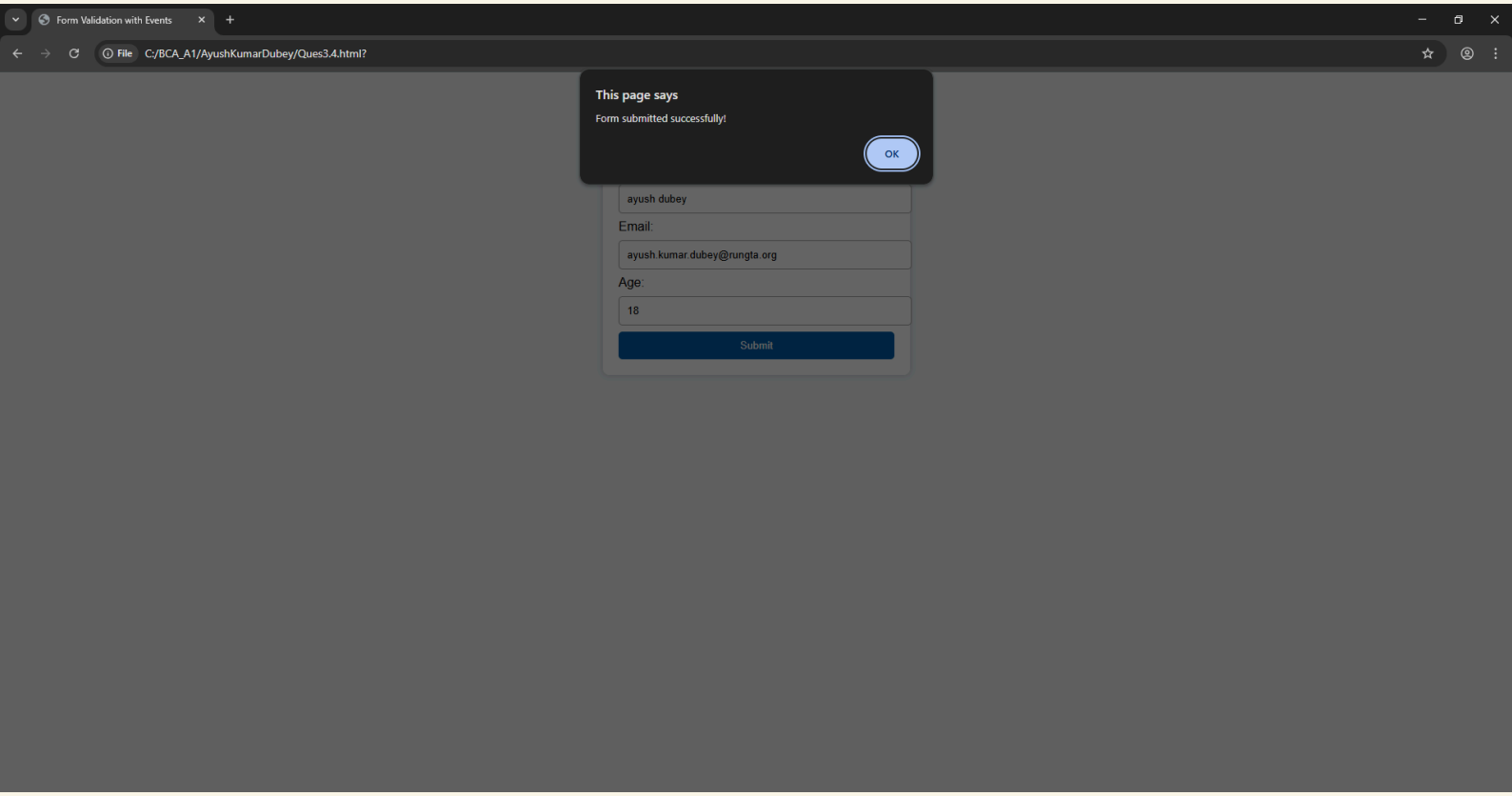
OUTPUT IMAGE



filled With User Data



FORM SUCCESSFULLY SUBMITTED EXAMPLE



Example

* Name must have 3 or more characters
* Email must match a proper format
* All fields are required
* Alerts appear when something is wrong

Conclusion

JavaScript events such as oninput and onsubmit provide powerful tools for building interactive form validation. Real-time feedback improves user experience, while final submit validation ensures accurate and clean data. Using alert() gives clear feedback before submitting the form.