TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

LAB REPORT

JS Lab Assignment

Submitted by:
Name: Chandan Kumar Shah
Roll No: 080BCT023
Department: III/I

Submitted to:
Department of Electronics
and Computer Engineering

Contents

1 Assignment 1: Smart Calculator with History

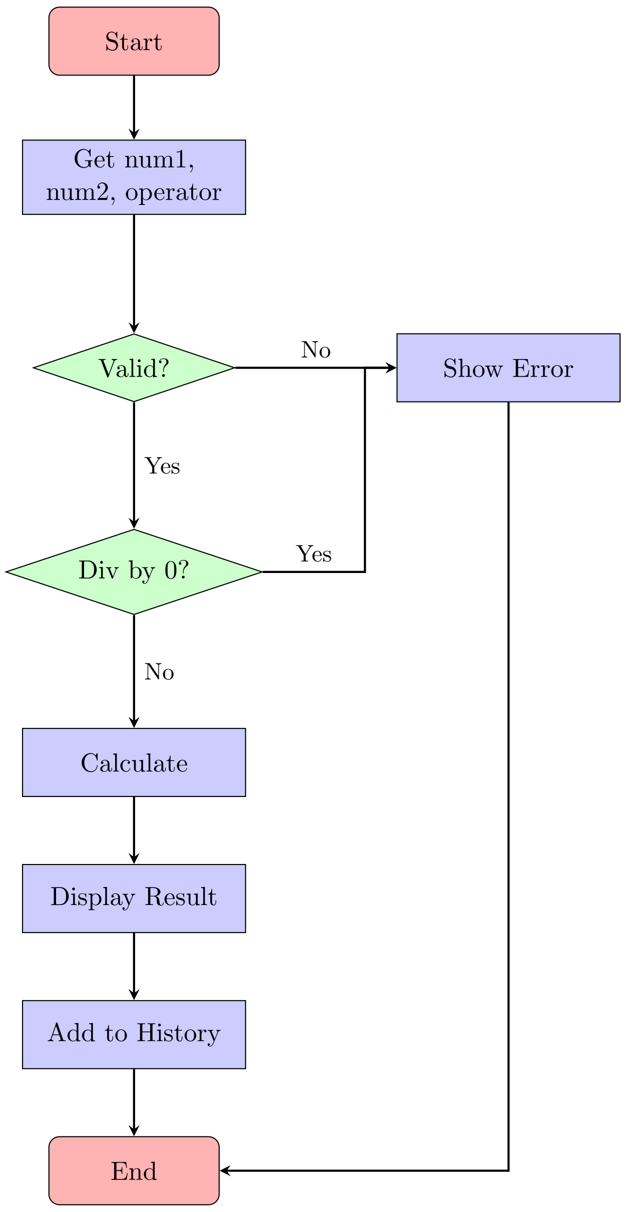
1.1 Problem Understanding

Design a calculator that performs basic arithmetic operations (addition, subtraction, multiplication, division), stores the last 5 calculations in history, displays calculation history dynamically, and allows clearing history. The calculator must handle invalid inputs such as division by zero.

1.2 List of JavaScript Concepts Used

- **Variables (let):** For storing numbers, results, and history array
- **Arrays:** To store calculation history (push, pop, unshift)
- **Functions:** calculate(), showResult(), addToHistory(), displayHistory(), clearHistory()
- **Switch Statement:** To handle different arithmetic operators
- **DOM Manipulation:** getElementById(), textContent, innerHTML
- **Input Validation:** isNaN() to check for valid numbers
- **Template Literals:** For string formatting with ${}
- **Array Methods:** forEach() for looping through history

1.3 Flow Diagram



1.4 Source Code

```
// This variable stores the history of calculations.
// An array is like a list that can hold multiple values.
let history = [];

// Function to add numbers or operators to the display
function appendToDisplay(value) {
    // Get the display element (the input box)
    const display = document.getElementById('display');

    // Add the clicked value to the current text in the display
    display.value += value;
}

// Function to clear the entire display
```

```javascript
function clearDisplay() {
    document.getElementById(’display’).value = ’’;
}

// Function to delete the last character
function deleteLast() {
    const display = document.getElementById(’display’);
    // .slice(0, -1) removes the last character from the string
    display.value = display.value.slice(0, -1);
}

// Function to calculate the result
function calculateResult() {
    const display = document.getElementById(’display’);
    const expression = display.value;

    try {
        // eval() takes a string of math (like "2+2") and calculates it.
        // WARNING: In real professional apps, eval() can be dangerous,
        // but for a simple student lab calculator, it is fine.
        const result = eval(expression);

        // Update the display with the result
        display.value = result;

        // Add this calculation to our history list
        addToHistory(expression + " = " + result);

    } catch (error) {
        // If there is an error (like dividing by zero or bad syntax), show "Error"
        display.value = ’Error’;
    }
}

// Function to add a calculation to the history array
function addToHistory(entry) {
    // Add the new entry to the BEGINNING of the array using unshift()
    history.unshift(entry);

    // If we have more than 5 items, remove the oldest (the last one)
    if (history.length > 5) {
        history.pop();
    }

    // Update the screen to show the new history list
    updateHistoryUI();
}

// Function to display the history on the screen
function updateHistoryUI() {
    const historyList = document.getElementById(’historyList’);

    // Clear the current list HTML so we don’t duplicate items
    historyList.innerHTML = ’’;

    // Loop through each item in our history array
    for (let i = 0; i < history.length; i++) {
        const item = history[i];

        // Create a new list item (<li>) tag
        const li = document.createElement('li');
        li.textContent = item; // Set the text inside the list item

        // Add the list item to the history list (<ul>)
        historyList.appendChild(li);
    }
}

// Function to clear the history
function clearHistory() {
    history = []; // Empty the array
    updateHistoryUI(); // Update the screen
}
```

Listing 1: Calculator Core Functions

1.5 Output Screenshots

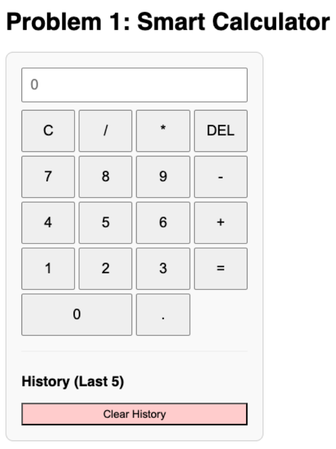


Figure 1: Smart Calculator with History

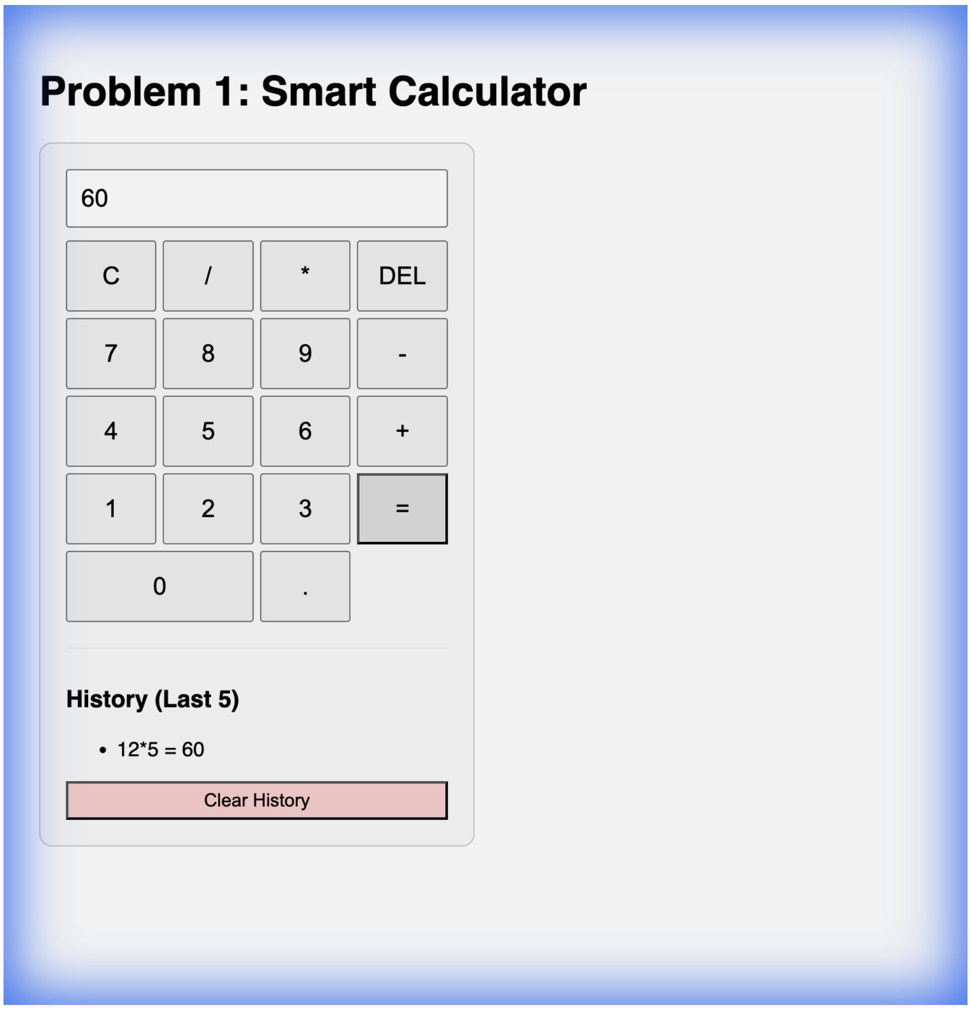


Figure 2: Calculator displaying calculation history

2 Assignment 2: Student Marks & Grade Analyzer

2.1 Problem Understanding

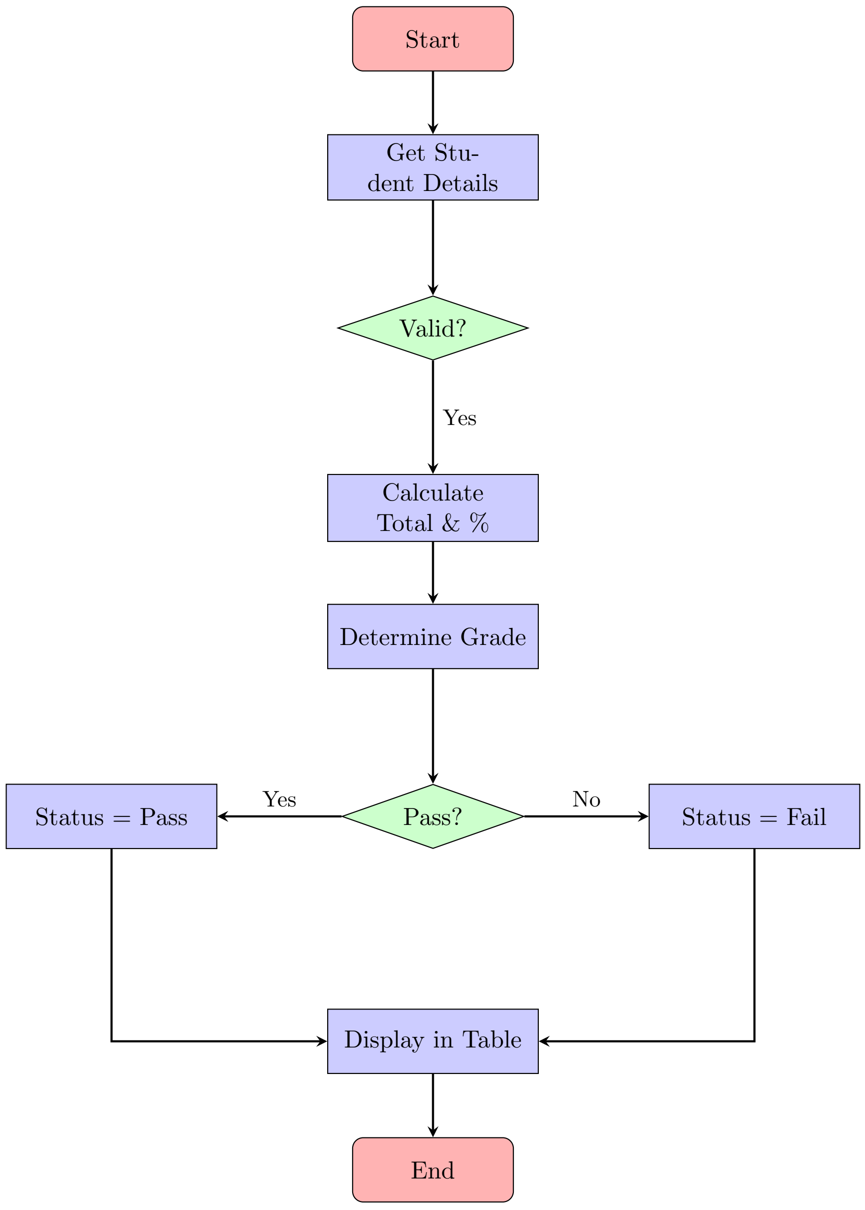
Create a system where student details are stored as objects, marks for multiple subjects are entered, total, percentage, and grade are calculated, results are displayed in a table, and pass/fail rows are highlighted using CSS via JavaScript.

2.2 List of JavaScript Concepts Used

- **Objects:** To store student data (name, rollNo, marks, grade)
- **Array of Objects:** students[] array holding multiple student objects
- **Functions:** addStudent(), calculateGrade(), displayResults(), clearForm()
- **If-Else Statements:** For grade calculation based on percentage
- **DOM Manipulation:** createElement(), appendChild(), innerHTML
- **CSS Class Manipulation:** classList.add() for pass/fail highlighting

- **parseInt():** Convert string input to integer
- **Ternary Operator:** For pass/fail status determination

2.3 Flow Diagram

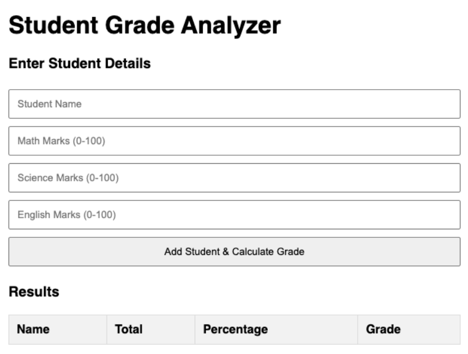


2.4 Source Code

```
// This array will hold all student objects
let students = [];

function addStudent() {
    // 1. Get values from the input fields
    const name = document.getElementById('name').value;
    // We use parseFloat to convert the text "95" to the number 95
    const math = parseFloat(document.getElementById('math').value);
    const science = parseFloat(document.getElementById('science').value
        );
    const english = parseFloat(document.getElementById('english').value
       );

    // Simple validation: Check if fields are empty
    if (name === "" || isNaN(math) || isNaN(science) || isNaN(english))
        {
        alert("Please fill in all fields correctly.");
        return; // Stop the function here
    }

    // 2. Calculate Total and Percentage
    const total = math + science + english;
    // (Total / Max Marks) * 100. Max marks is 300 (3 subjects * 100)
    const percentage = (total / 300) * 100;

    // 3. Determine Grade using if-else
    let grade = "";
    if (percentage >= 90) {
        grade = "A";
    } else if (percentage >= 80) {
        grade = "B";
    } else if (percentage >= 70) {
        grade = "C";
    } else if (percentage >= 60) {
        grade = "D";
    } else {
        grade = "F";
    }

    // 4. Create a student object
    const student = {
        name: name,
        total: total,
        percentage: percentage.toFixed(2), // toFixed(2) keeps only 2
            decimal places (e.g., 85.50)
        grade: grade
    };

    // Add to our list
    students.push(student);

    // 5. Add a new row to the table
    addStudentToTable(student);

    // Clear inputs for next student
    document.getElementById('name').value = '';
    document.getElementById('math').value = '';
    document.getElementById('science').value = '';
    document.getElementById('english').value = '';
}

function addStudentToTable(student) {
    const tableBody = document.querySelector('#resultTable tbody');

    // Create a new table row <tr>
    const row = document.createElement('tr');

    // Add class for styling if passed or failed
    if (student.grade === "F") {
        row.classList.add('fail'); // Red color
    } else {
        row.classList.add('pass'); // Green color
    }

    // Insert the cells with student data
    // innerHTML is easier for beginners than creating many 'td' elements manually
    row.innerHTML = `
        <td>${student.name}</td>
        <td>${student.total}</td>
        <td>${student.percentage}%</td>
        <td>${student.grade}</td>
    `;

    // Add the row to the table body
    tableBody.appendChild(row);
}
```

Listing 2: Student Grade Analyzer Functions

2.5 Output Screenshots



Student Grade Analyzer

Enter Student Details

Student Name

Math Marks (0-100)

Science Marks (0-100)

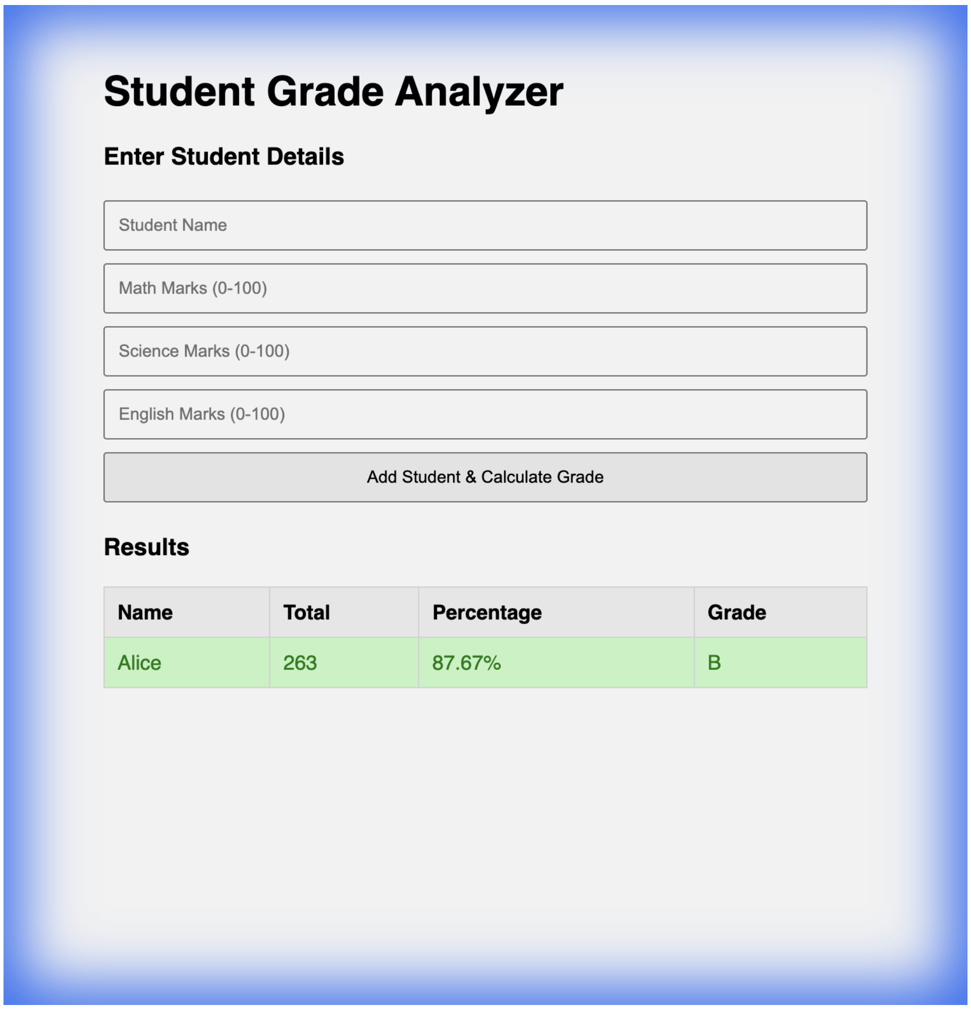
English Marks (0-100)

Add Student & Calculate Grade

Results

Name	Total	Percentage	Grade

Figure 3: Student Marks & Grade Analyzer



Name	Total	Percentage	Grade
Alice	263	87.67%	B

Figure 4: Grade Analyzer with student data added

3 Assignment 3: Dynamic To-Do List with Status Filter

3.1 Problem Understanding

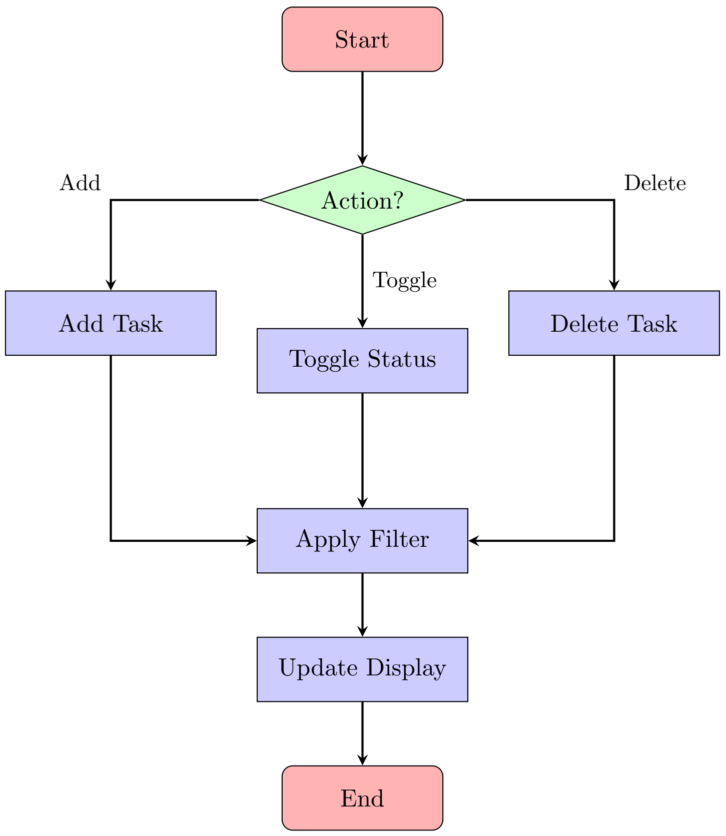
Build a To-Do application that allows users to add, delete, and mark tasks as completed, filter tasks by status (All/Completed/Pending), and display count of completed tasks.

3.2 List of JavaScript Concepts Used

- **Arrays:** To store task objects
- **Objects:** Task objects with id, text, completed properties
- **Date.now():** Generate unique IDs for tasks
- **filter() Method:** Filter tasks by completion status
- **Boolean Toggle:** !task.completed to toggle status
- **Event Listeners:** For keyboard input (Enter key)

- **Template Literals:** Dynamic HTML generation

3.3 Flow Diagram



3.4 Source Code

```javascript
let tasks = [];

function addTask() {
    const taskInput = document.getElementById('taskInput');
    const taskText = taskInput.value;

    if (taskText === '') {
        alert("Please enter a task!");
        return;
    }

    // Create a task object
    const task = {
        id: Date.now(), // Unique ID based on time
        text: taskText,
        completed: false
    };

    tasks.push(task);
    taskInput.value = ''; // Clear input

    renderTasks(); // Update the screen
}
```

```

function toggleComplete(id) {
    // Find the task in the array
    for (let i = 0; i < tasks.length; i++) {
        if (tasks[i].id === id) {
            // Flip the completed status (true becomes false, false becomes true)
            tasks[i].completed = !tasks[i].completed;
            break;
        }
    }
    renderTasks();
}

function deleteTask(id) {
    // Filter out the task with the given ID
    // keeping only tasks where task.id is NOT equal to the id we want to delete
    tasks = tasks.filter(task => task.id !== id);
    renderTasks();
}

let currentFilter = 'all';

function filterTasks(status) {
    currentFilter = status;
    renderTasks();
}

function renderTasks() {
    const taskList = document.getElementById('taskList');
    taskList.innerHTML = ''; // Clear current list

    let completedCount = 0;

    for (let i = 0; i < tasks.length; i++) {
        const task = tasks[i];

        // Filter logic
        if (currentFilter === 'active' && task.completed) continue; // Skip completed if viewing active
        if (currentFilter === 'completed' && !task.completed) continue; // Skip active if viewing completed

        // Count completed tasks
        if (task.completed) completedCount++;

        // Create list item
        const li = document.createElement('li');
        if (task.completed) {
            li.classList.add('completed');
        }

        // Add HTML for the task item
        li.innerHTML = `
            <span onclick="toggleComplete(${task.id})" style="cursor: pointer;">${task.text}</span>
            <button onclick="deleteTask(${task.id})" style="background:
                #ffcccc; border: none; color: red;">X</button>
        `;

        taskList.appendChild(li);
    }

    // Update count
    document.getElementById('completedCount').innerText =
        completedCount;
}
```

Listing 3: To-Do List Core Functions

3.5 Output Screenshots

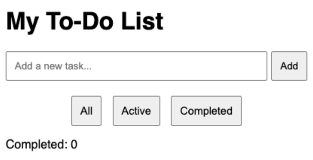


Figure 5: Dynamic To-Do List with Status Filter

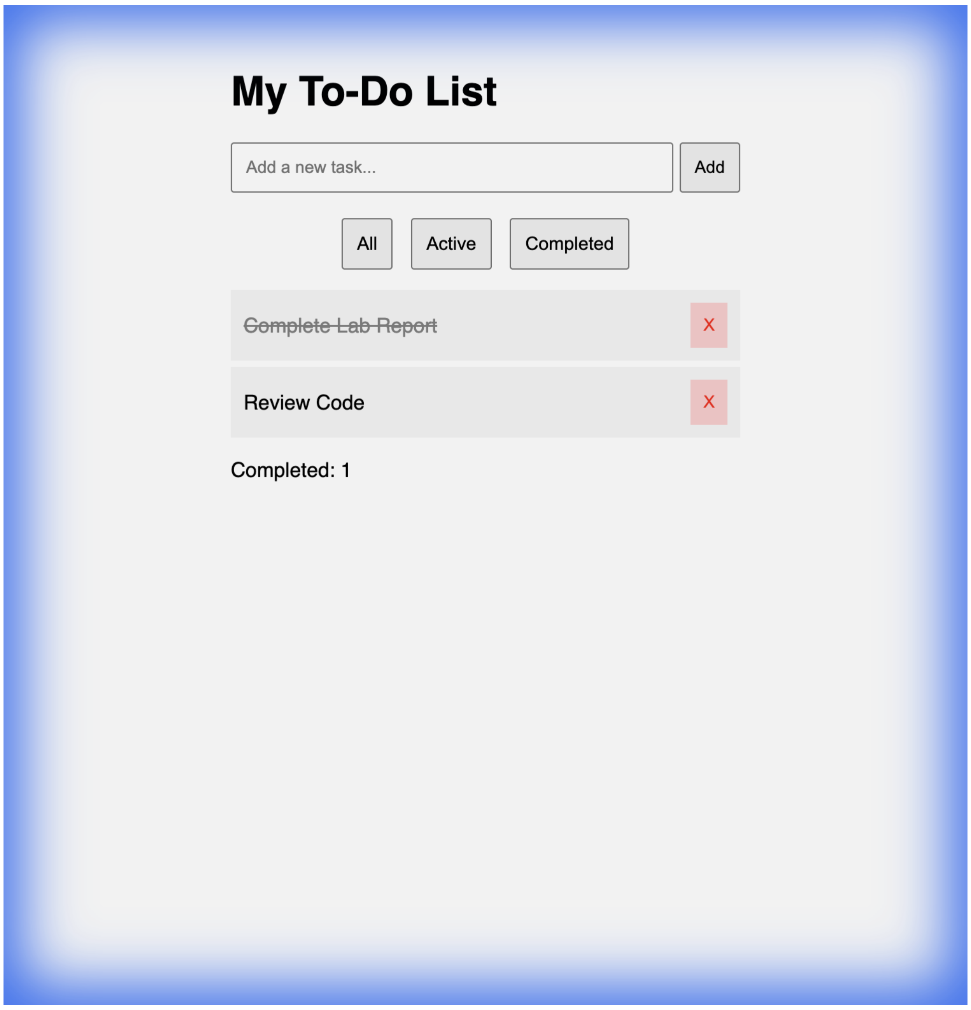


Figure 6: To-Do List showing active and completed tasks

4 Assignment 4: Number Guessing Game

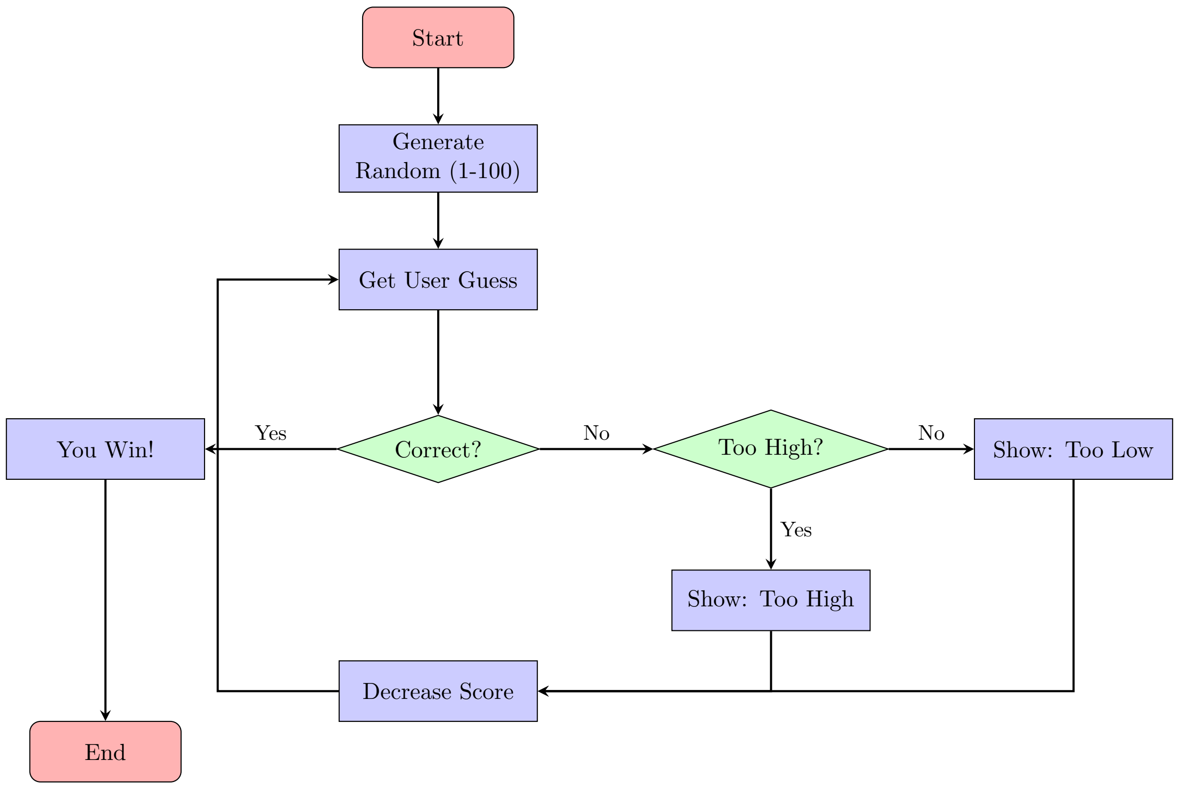
4.1 Problem Understanding

Develop a game where the system generates a random number between 1-100, user guesses the number, feedback is shown (High/Low/Correct), and score decreases on wrong attempts.

4.2 List of JavaScript Concepts Used

- **Math.random():** Generate random number
- **Math.floor():** Round down to integer
- **Comparison Operators:** ===, ¿, ¡ for guess checking
- **Boolean Variables:** gameOver flag
- **Score Tracking:** Decrement score on wrong guesses
- **Input Validation:** Range checking (1-100)

4.3 Flow Diagram

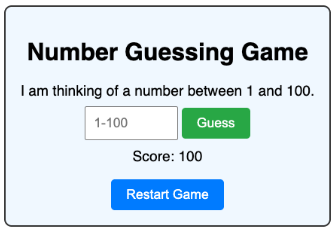


4.4 Source Code

```
let secretNumber;
let score = 100;
let attempts = 0;
let gameOver = false;

function initGame() {
    secretNumber = Math.floor(Math.random() * 100) + 1;
    score = 100;
    attempts = 0;
    gameOver = false;
}

function checkGuess() {
    if (gameOver) return;

    let guess = parseInt(document.getElementById('guessInput').value);
    if (isNaN(guess) || guess < 1 || guess > 100) {
        showFeedback('Enter number 1-100', 'hint');
        return;
    }

    attempts++;

    if (guess === secretNumber) {
        showFeedback('Correct! You win!', 'correct');
        gameOver = true;
    } else if (guess > secretNumber) {
        showFeedback('Too High!', 'high');
        score -= 10;
    } else {
        showFeedback('Too Low!', 'low');
        score -= 10;
    }

    if (score <= 0) gameOver = true;
    updateDisplay();
}
```

Listing 4: Number Guessing Game Functions

4.5 Output Screenshots



Number Guessing Game

I am thinking of a number between 1 and 100.

Score: 100

Figure 7: Number Guessing Game

5 Assignment 5: Interactive Quiz Application

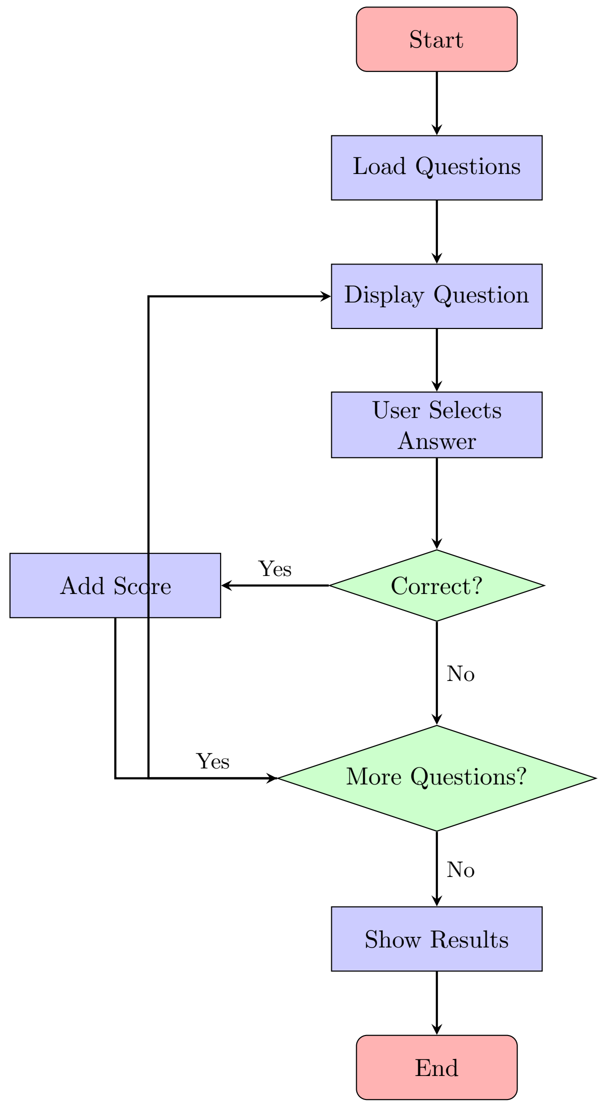
5.1 Problem Understanding

Create a quiz app where questions are stored as objects, one question is shown at a time, user selects an answer, and score is calculated and shown at the end.

5.2 List of JavaScript Concepts Used

- **const:** For immutable questions array
- **Array of Objects:** Questions with options and correct index
- **Array Index Access:** questions[currentQuestionIndex]
- **String.fromCharCode():** Generate A, B, C, D labels
- **querySelectorAll():** Select multiple elements
- **classList Manipulation:** For correct/wrong styling

5.3 Flow Diagram

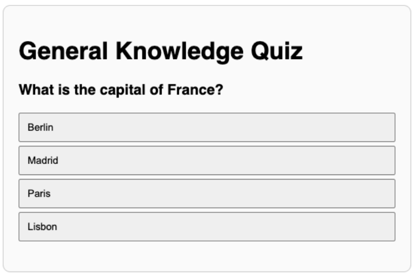


5.4 Source Code

```
const questions = [
    {
        question: "What does HTML stand for?",
        options: ["Hyper Text Markup Language", "High Tech",
                  "Hyper Transfer", "Home Tool"],
        correct: 0
    }
];

let currentQuestionIndex = 0;
let score = 0;

function displayQuestion() {
    let current = questions[currentQuestionIndex];
    document.getElementById('questionText').textContent =
        current.question;

    let html = '';
    current.options.forEach(function(option, index) {
        let letter = String.fromCharCode(65 + index);
        html += `<button class="option"
                    onclick="selectAnswer(${index})">
                    ${letter}. ${option}</button>`;
    });
    document.getElementById('optionsContainer').innerHTML = html;
}

function selectAnswer(index) {
    let correct = questions[currentQuestionIndex].correct;
    if (index === correct) score++;

    let options = document.querySelectorAll('.option');
    options[correct].classList.add('correct');
    if (index !== correct) {
        options[index].classList.add('wrong');
    }
}
```

Listing 5: Quiz Application Functions

5.5 Output Screenshots



General Knowledge Quiz

What is the capital of France?

Figure 8: Interactive Quiz Application

6 Assignment 6: Digital Clock & Countdown Timer

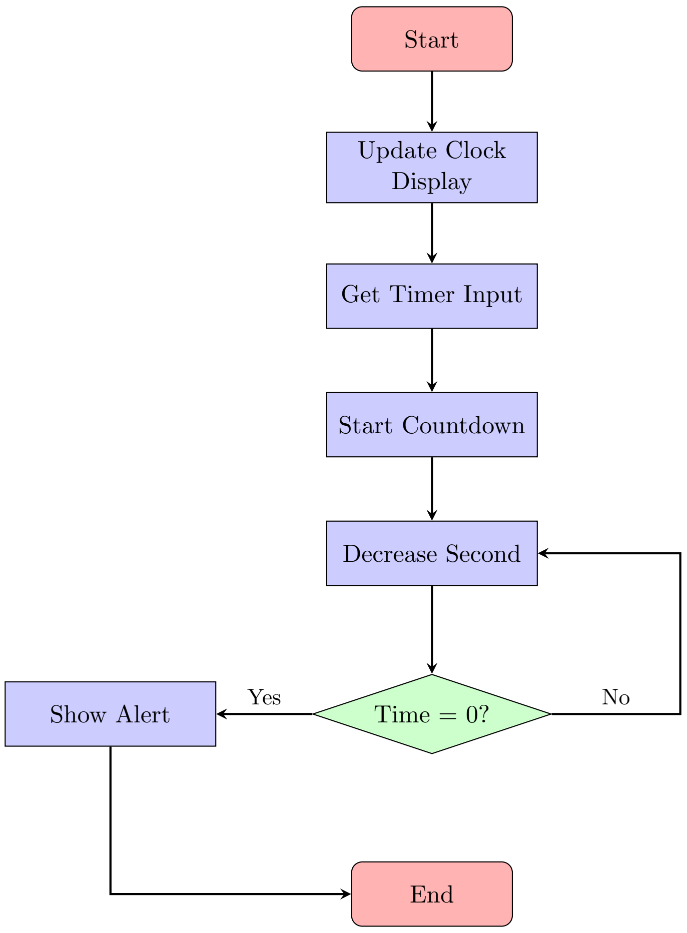
6.1 Problem Understanding

Create a real-time digital clock and a countdown timer that takes user input and alerts when time reaches zero.

6.2 List of JavaScript Concepts Used

- **Date Object:** new Date() for current time
- **Date Methods:** getHours(), getMinutes(), getSeconds()
- **setInterval():** Execute function repeatedly
- **clearInterval():** Stop the interval
- **padStart():** Format numbers with leading zeros
- **alert():** Show popup when timer ends

6.3 Flow Diagram



6.4 Source Code

```
// Digital Clock
function updateClock() {
    let now = new Date();
    let hours = String(now.getHours()).padStart(2, '0');
    let minutes = String(now.getMinutes()).padStart(2, '0');
    let seconds = String(now.getSeconds()).padStart(2, '0');

    document.getElementById('clockDisplay').textContent =
        `${hours}:${minutes}:${seconds}`;
}
setInterval(updateClock, 1000);

// Countdown Timer
let timerInterval = null;
let totalSeconds = 0;

function startTimer() {
    let hours = parseInt(document.getElementById('hours').value);
    let minutes = parseInt(document.getElementById('minutes').value);
    let seconds = parseInt(document.getElementById('seconds').value);

    totalSeconds = (hours * 3600) + (minutes * 60) + seconds;

    timerInterval = setInterval(function() {
        totalSeconds--;
        updateTimerDisplay();
        if (totalSeconds <= 0) {
            clearInterval(timerInterval);
            alert('Time is up!');
        }
    }, 1000);
}
```

Listing 6: Clock and Timer Functions

6.5 Output Screenshots

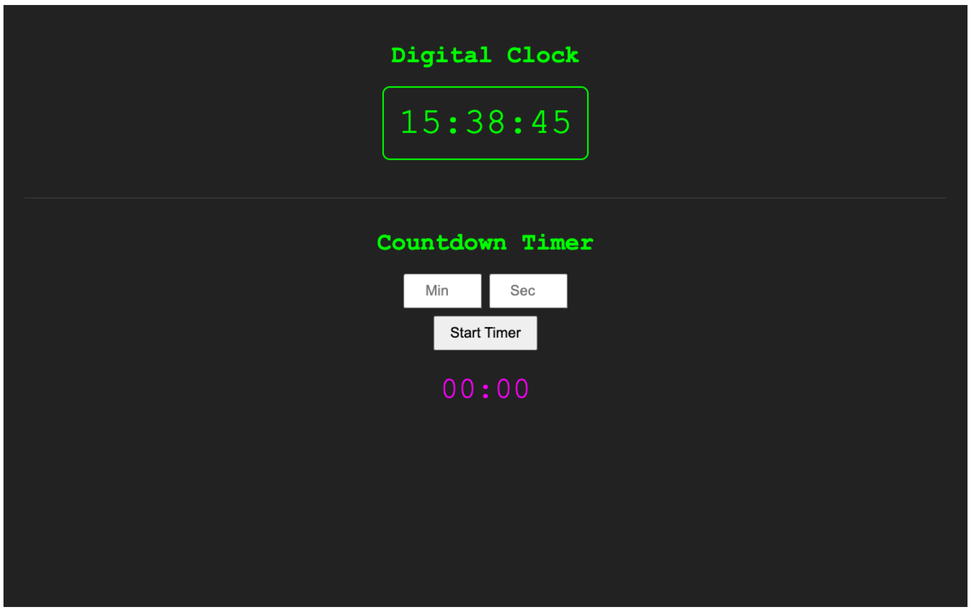


Figure 9: Digital Clock & Countdown Timer

7 Assignment 7: Dynamic Table Generator

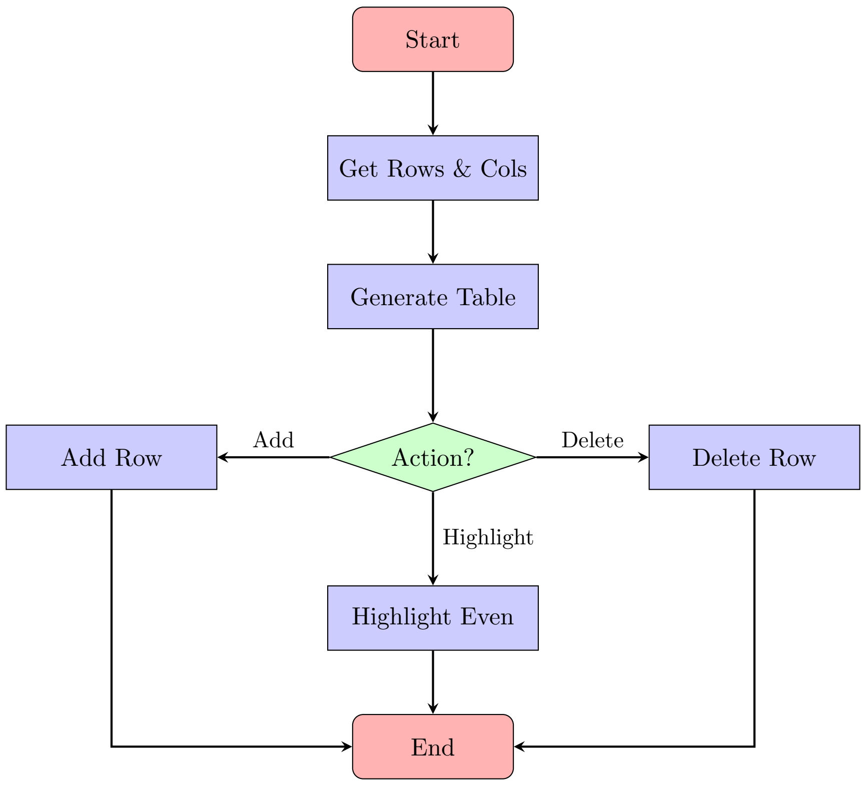
7.1 Problem Understanding

Design a program that takes number of rows and columns as input, generates a table dynamically, and adds buttons to add row, delete row, and highlight even rows.

7.2 List of JavaScript Concepts Used

- **Nested Loops:** For creating rows and columns
- **createElement():** Create HTML elements dynamically
- **appendChild():** Add elements to DOM
- **getElementsByTagName():** Select elements by tag
- **Modulo Operator (%):** Check for even/odd rows
- **lastElementChild:** Access last child element
- **removeChild():** Remove elements from DOM

7.3 Flow Diagram



7.4 Source Code

```javascript
function generateTable() {
    const rows = document.getElementById('rows').value;
    const cols = document.getElementById('cols').value;
    const container = document.getElementById('tableContainer');

    // Create a new table element
    const table = document.createElement('table');

    // Loop to create rows
    for (let i = 0; i < rows; i++) {
        const tr = document.createElement('tr'); // Table Row

        // Loop to create columns (cells) inside each row
        for (let j = 0; j < cols; j++) {
            const td = document.createElement('td'); // Table Data (
                Cell)
            td.innerText = "Row " + (i + 1) + ", Col " + (j + 1);
            tr.appendChild(td);
        }

        table.appendChild(tr);
    }

    // Clear previous table and add the new one
    container.innerHTML = '';
    container.appendChild(table);
}

```

```javascript
function highlightEven() {
    // Get all rows in the table
    const rows = document.querySelectorAll('table tr');

    for (let i = 0; i < rows.length; i++) {
        // Highlight 2nd, 4th, 6th... rows (indices 1, 3, 5...)
        if (i % 2 !== 0) {
            rows[i].classList.toggle('even-row');
        }
    }
}

function addRow() {
    const table = document.querySelector('table');

    if (!table) {
        alert("Please generate a table first!");
        return;
    }

    // Insert a new row at the end
    const newRow = table.insertRow();

    // Check how many columns the first row has
    // If table has no rows, fallback to 3 columns (or user input)
    let colCount = 3;
    if (table.rows.length > 1) { // checking >1 because we just added
        one, so >0 is always true
        colCount = table.rows[0].cells.length;
    } else {
        // If we deleted all rows and added one back, use the input
            value
        colCount = document.getElementById('cols').value;
    }

    const rowIndex = table.rows.length; // Current row count

    for (let i = 0; i < colCount; i++) {
        const cell = newRow.insertCell();
        cell.innerText = "Row " + rowIndex + ", Col " + (i + 1);
    }
}

function deleteRow() {
    const table = document.querySelector('table');

    if (!table || table.rows.length === 0) {
        alert("No rows to delete!");
        return;
    }

    // Delete the last row
    table.deleteRow(table.rows.length - 1);
}
```

Listing 7: Dynamic Table Functions

7.5 Output Screenshots

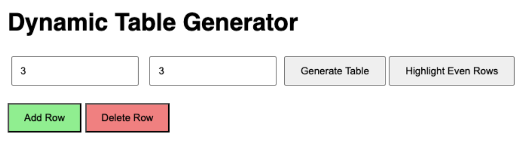
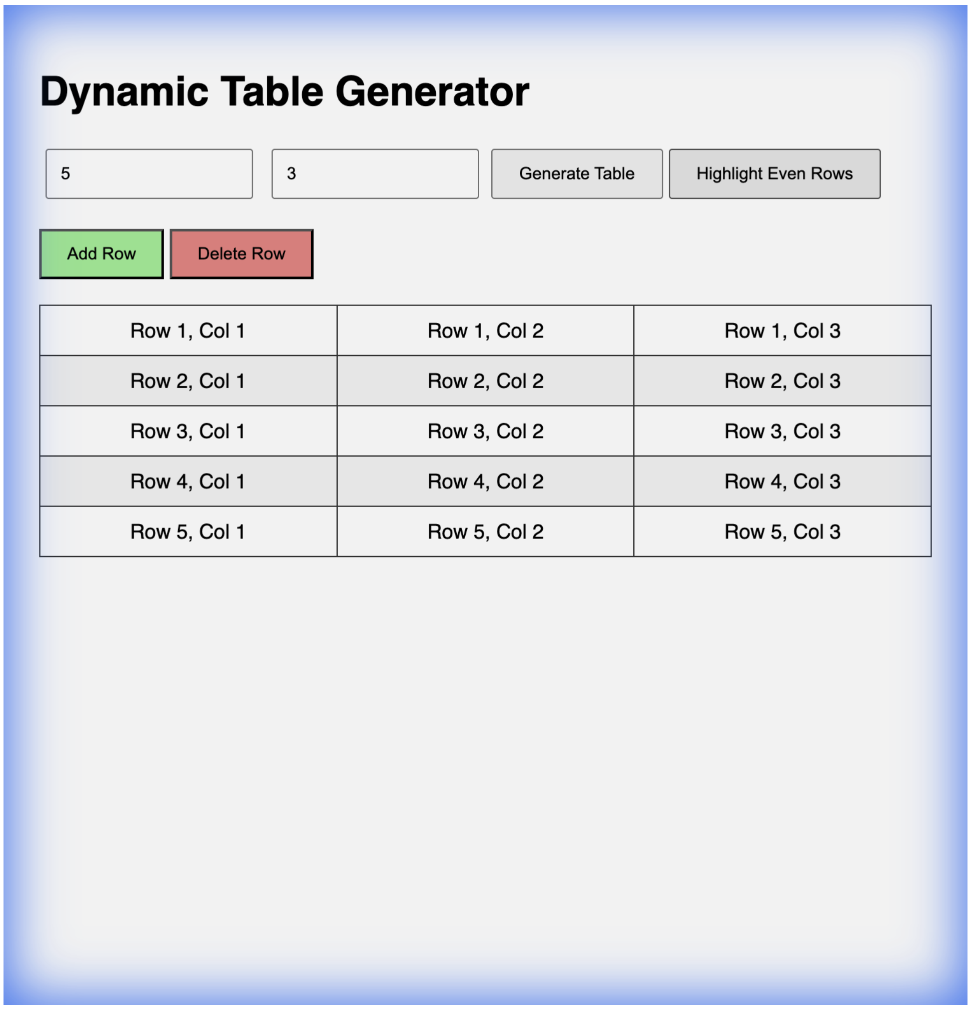


Figure 10: Dynamic Table Generator



Row 1, Col 1	Row 1, Col 2	Row 1, Col 3
Row 2, Col 1	Row 2, Col 2	Row 2, Col 3
Row 3, Col 1	Row 3, Col 2	Row 3, Col 3
Row 4, Col 1	Row 4, Col 2	Row 4, Col 3
Row 5, Col 1	Row 5, Col 2	Row 5, Col 3

Figure 11: Dynamic Table with highlighted even rows

8 Assignment 8: Simple E-Commerce Cart

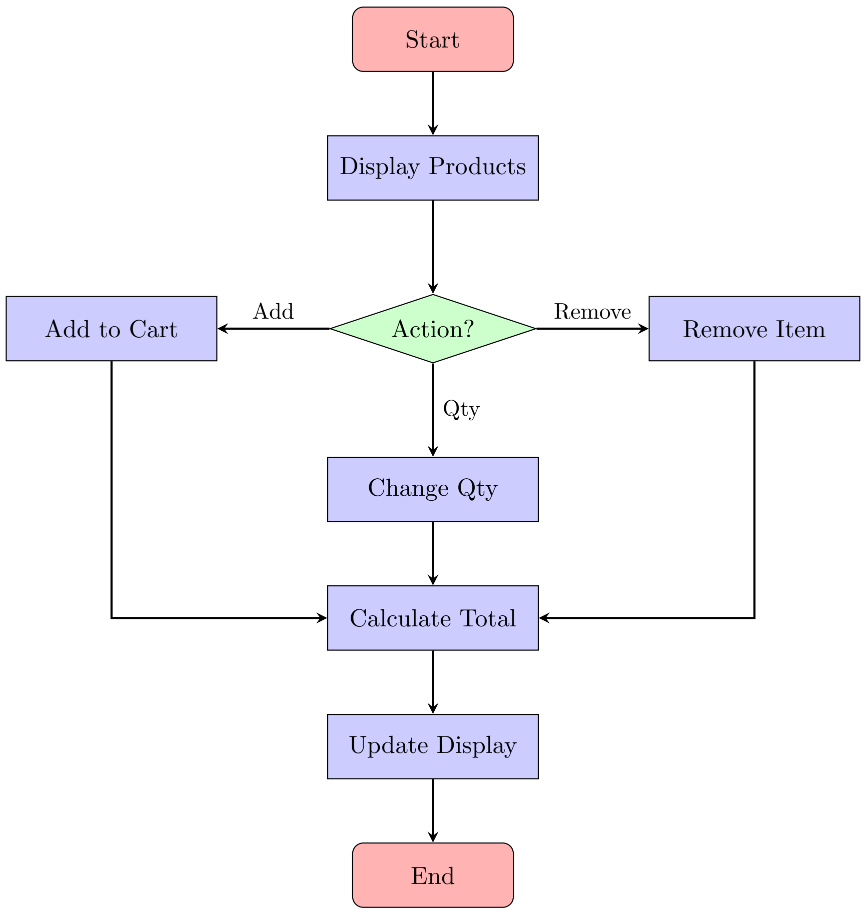
8.1 Problem Understanding

Create a shopping cart where products are stored as objects, user can add/remove items, total price updates dynamically, and quantity changes affect total.

8.2 List of JavaScript Concepts Used

- **Array of Product Objects:** Store product data
- **find() Method:** Locate product by ID
- **toFixed():** Format currency to 2 decimal places
- **filter() Method:** Remove items from cart
- **Accumulator Pattern:** Calculate total price
- **Dynamic Updates:** Real-time cart total calculation

8.3 Flow Diagram

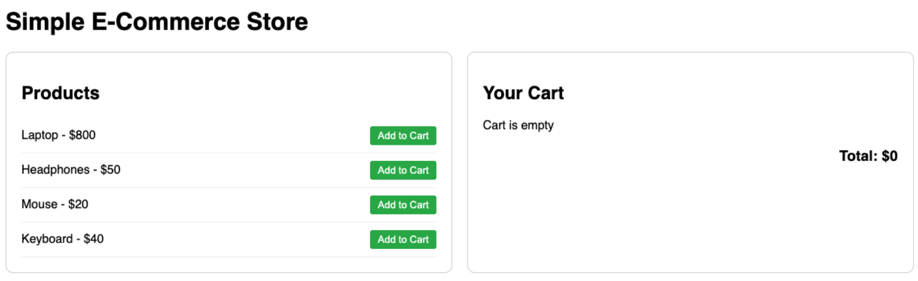


8.4 Source Code

```
const products = [
    { id: 1, name: 'Laptop', price: 999.99, emoji: '      ' },
    { id: 2, name: 'Phone', price: 699.99, emoji: '      ' }
];

let cart = [];

function addToCart(productId) {
    let product = products.find(p => p.id === productId);
    let existing = cart.find(item => item.id === productId);

    if (existing) {
        existing.quantity++;
    } else {
        cart.push({
            id: product.id, name: product.name,
            price: product.price, quantity: 1
        });
    }
    updateCartDisplay();
}

function calculateTotal() {
    let total = 0;
    cart.forEach(function(item) {
        total += item.price * item.quantity;
    });
    return total;
}

function removeFromCart(productId) {
    cart = cart.filter(item => item.id !== productId);
    updateCartDisplay();
}
```

Listing 8: Shopping Cart Functions

8.5 Output Screenshots



Simple E-Commerce Store

Products

Laptop - $800

Headphones - $50

Mouse - $20

Keyboard - $40

Your Cart

Cart is empty

Total: $0

Figure 12: Simple E-Commerce Cart

9 Assignment 9: Form Validation System

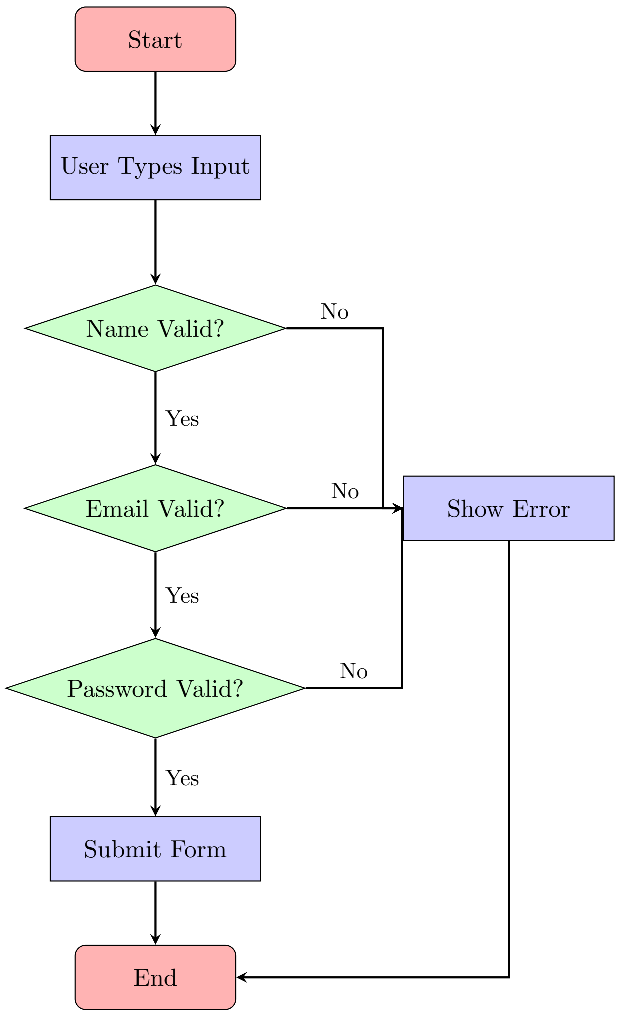
9.1 Problem Understanding

Develop a form that validates name, email, and password fields, shows error messages dynamically, and prevents submission if validation fails.

9.2 List of JavaScript Concepts Used

- **trim() Method:** Remove whitespace from inputs
- **Regular Expressions:** Email pattern validation
- **test() Method:** Check if string matches regex
- **oninput Event:** Real-time validation as user types
- **preventDefault():** Stop form default submission
- **classList Manipulation:** Show/hide error messages

9.3 Flow Diagram



9.4 Source Code

```
let isNameValid = false;
let isEmailValid = false;
let isPasswordValid = false;

function validateName() {
    let name = document.getElementById('name').value.trim();
    let errorDiv = document.getElementById('nameError');

    if (name.length >= 3) {
        isNameValid = true;
        errorDiv.classList.remove('show');
    } else {
        isNameValid = false;
        errorDiv.classList.add('show');
    }
}

function validateEmail() {
    let email = document.getElementById('email').value.trim();
    let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    isEmailValid = emailPattern.test(email);
}

function validateForm(event) {
    event.preventDefault();
    validateName();
    validateEmail();
    validatePassword();

    if (isNameValid && isEmailValid && isPasswordValid) {
        // Form is valid - submit
        showSuccess();
    } else {
        alert('Please fix errors');
    }
}
```

Listing 9: Form Validation Functions

9.5 Output Screenshots

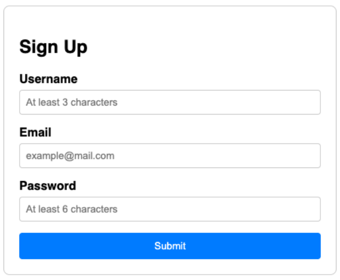


Figure 13: Form Validation System

10 Assignment 10: Color & Theme Manager

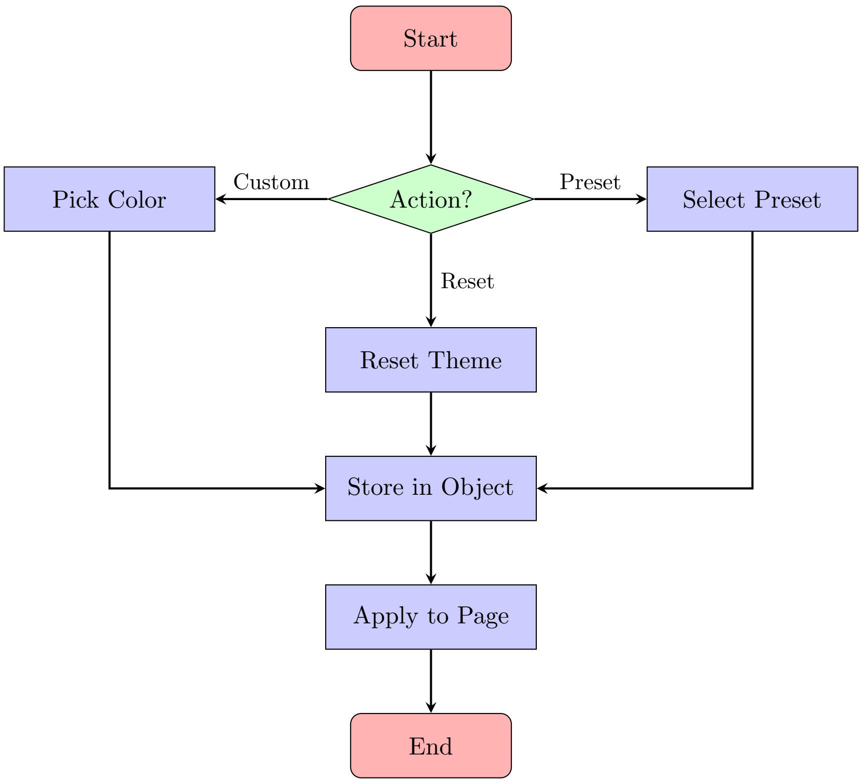
10.1 Problem Understanding

Create a webpage where user selects theme colors, background/text/button colors change dynamically, selected theme is stored in an object, and reset option restores default theme.

10.2 List of JavaScript Concepts Used

- **Objects:** Store current theme and preset themes
- **Color Input:** HTML5 color picker input type
- **style Property:** Modify CSS via JavaScript
- **Object Properties:** Access/modify theme values
- **Event Handling:** onchange for color picker
- **Preset Themes:** Object containing multiple theme configurations

10.3 Flow Diagram



10.4 Source Code

```javascript
// This object stores our current theme settings
let currentTheme = {
    background: 'white',
    text: 'black'
};

function applyTheme() {
    // 1. Get values from dropdowns
    const bgSelect = document.getElementById('bgColor').value;
    const textSelect = document.getElementById('textColor').value;

    // 2. Apply styles to the Body element
    document.body.style.backgroundColor = bgSelect;
    document.body.style.color = textSelect;

    // 3. Update our theme object
    currentTheme.background = bgSelect;
    currentTheme.text = textSelect;

    // 4. Log the change
    displayThemeStatus();
}

function resetTheme() {
    // Restore defaults
    document.body.style.backgroundColor = 'white';
    document.body.style.color = 'black';

    // Reset dropdowns
    document.getElementById('bgColor').value = 'white';
    document.getElementById('textColor').value = 'black';

    // Update object
    currentTheme = {
        background: 'white',
        text: 'black'
    };

    displayThemeStatus();
}

function displayThemeStatus() {
    const log = document.getElementById('currentThemeLog');
    // Using JSON.stringify makes it easy to see the object info as
        text
    log.innerText = "Current Theme Object:\n" + JSON.stringify(
        currentTheme, null, 2);
}
```

Listing 10: Theme Manager Functions

10.5 Output Screenshots

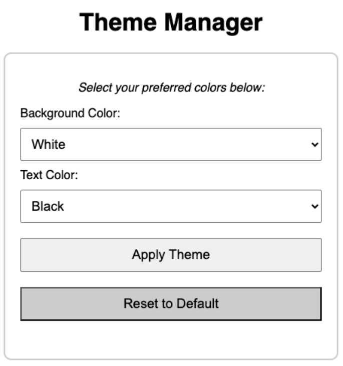
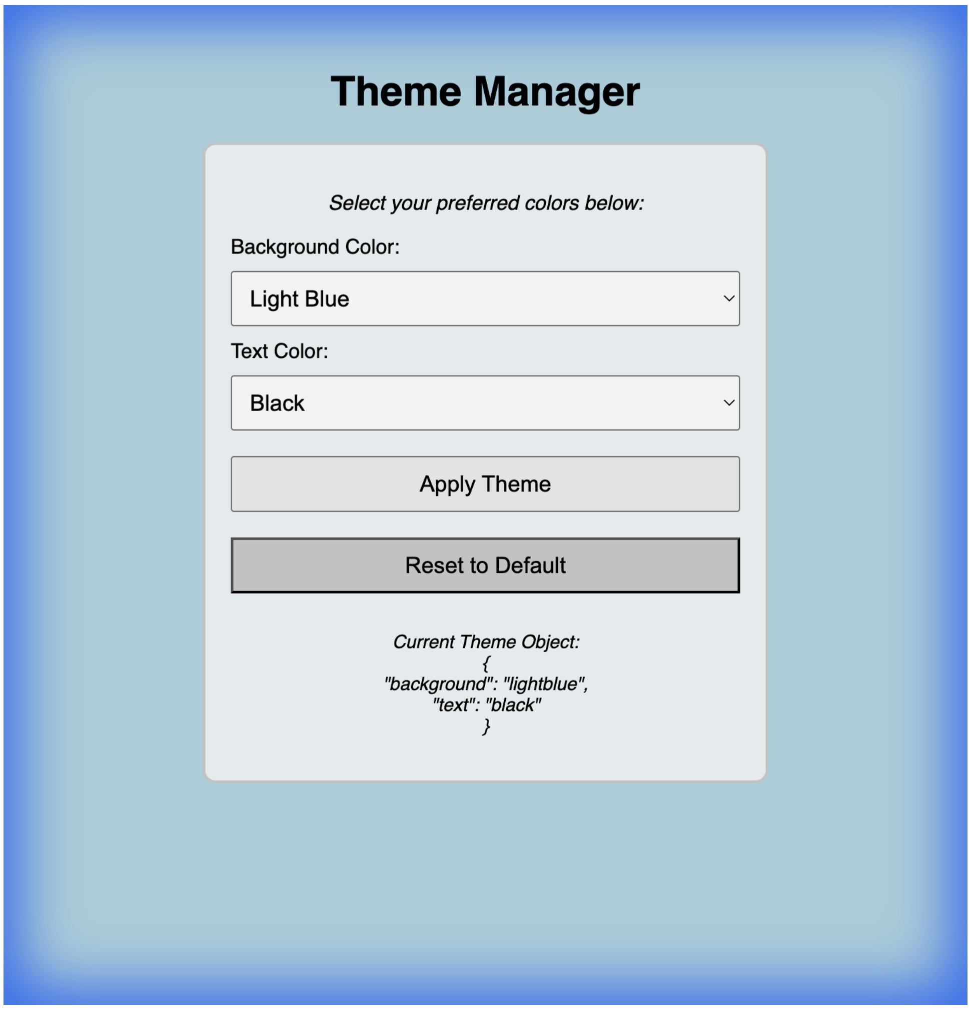


Figure 14: Color & Theme Manager



```
{
"background": "lightblue",
"text": "black"
}
```

Figure 15: Theme Manager with blue background applied

11 Conclusion

In this lab, we successfully demonstrated the implementation of 10 different JavaScript applications using Vanilla JavaScript (without any external libraries or frameworks). Key concepts we have learned include:

- DOM manipulation for dynamic content updates
- Event handling for user interactions
- Array methods (push, pop, filter, find, forEach)
- Object-oriented data storage
- Form validation using regular expressions
- Timer functions (setInterval, clearInterval)
- CSS manipulation through JavaScript

We implemented all applications with clean, readable code and proper error handling to ensure robust functionality.