Contents

UNIT III:

UNIT III: Advanced CSS – Layouts, Box Model and Visual Styling Color and Background Styling, Font and Text Styling, CSS Box Model, Positioning and Background Effects

Experiments:

1. Demonstrate different color definitions in CSS (hex, RGB, names).
2. Use CSS to place a background image fixed at the center and repeat horizontally.
3. Style text with font size, weight, style, alignment, transformation, and decoration.
4. Illustrate the CSS Box Model using content, padding, border, and margin differences.

UNIT III: Advanced CSS – Layouts, Box Model and Visual Styling

(Color & Background Styling, Font & Text Styling, CSS Box Model, Positioning & Background Effects)

3.1. CSS Layouts

Layouts define how elements are arranged on a webpage.

a) Display Property

- block: Occupies full width (, <p>).
- inline: Occupies only required width (, <a>).
- inline-block: Behaves like inline but accepts box model.
- none: Hides element.

b) Flexbox Layout

- One-dimensional layout model (row or column).

```
.container {
  display: flex;
  justify-content: space-between; /* alignment along main axis */
  align-items: center; /* alignment along cross axis */
}
```

c) Grid Layout

- Two-dimensional layout model (rows + columns).

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  gap: 20px;
}
```

d) Float & Clear

- Old method for layouts, less used now.

```
img { float: left; }
p { clear: both; }
```

3.2. CSS Box Model

Every element is represented as a rectangular box.

- **Content** → Text, images inside element.
- **Padding** → Space between content and border.
- **Border** → Surrounds padding + content.
- **Margin** → Space outside the border.

Example:

```
div {
  width: 200px;
```

```
  padding: 15px;
  border: 2px solid black;
  margin: 10px;
}
```

Total Width = Content + Padding + Border + Margin

3.3. Color and Background Styling

a) Color

- Named colors → red, blue.
- Hex values → #ff0000.
- RGB → rgb(255,0,0).
- RGBA → rgba(255,0,0,0.5) (with transparency).

b) Backgrounds

- **Background Color**

```
body { background-color: lightgray; }
```

- **Background Image**

```
div { background-image: url("bg.jpg"); }
```

- **Background Repeat** → repeat, no-repeat, repeat-x, repeat-y.
- **Background Size** → auto, cover, contain.
- **Background Attachment** → scroll, fixed.

c) Gradient Backgrounds

- **Linear Gradient**

```
div { background: linear-gradient(to right, red, yellow); }
```

- **Radial Gradient**

```
div { background: radial-gradient(circle, blue, white); }
```

3.4. Font and Text Styling

Typography improves readability and design.

- font-family: Defines typeface (Arial, serif).
- font-size: Size of text (px, em, rem).
- font-weight: Thickness (normal, bold).
- font-style: Normal, italic, oblique.
- color: Text color.
- text-align: left, right, center, justify.
- text-transform: uppercase, lowercase, capitalize.
- text-decoration: underline, line-through, overline.
- letter-spacing & word-spacing: Adjust spacing.
- line-height: Vertical spacing.

Example:

```
h1 {
  font-family: "Times New Roman", serif;
  font-size: 28px;
  font-weight: bold;
  text-align: center;
  color: darkblue;
  letter-spacing: 2px;
}
```

3.5. Positioning in CSS

Controls how elements are placed on the page.

- **Static** (default, follows normal flow).
- **Relative** (offset from normal position).
- **Absolute** (relative to nearest positioned ancestor).
- **Fixed** (stays in viewport while scrolling).
- **Sticky** (relative until a scroll threshold, then sticks).

Example:

```
div { position: sticky; top: 0; background: yellow; }
```

3.6. Background Effects

Enhance web design with advanced effects:

- **Multiple Backgrounds**

```
div {
  background: url("stars.png"), linear-gradient(to bottom, blue, black);
}
```

- **Opacity**

```
div { background-color: rgba(0, 128, 0, 0.4); }
```

- **Box Shadow**

```
div { box-shadow: 5px 5px 15px rgba(0,0,0,0.5); }
```

- **Text Shadow**

```
h1 { text-shadow: 2px 2px 5px gray; }
```

3.7. Example – Styled Card

```
<div class="card">
  <h2>Hello CSS</h2>
  <p>This card demonstrates layouts, box model, text styling, and background effects.</p>
</div>
.card {
  width: 300px;
  padding: 20px;
  margin: 30px auto;
  border: 2px solid #333;
  background: linear-gradient(to bottom, #f9f9f9, #ddd);
  font-family: Arial, sans-serif;
  text-align: center;
  box-shadow: 0px 4px 8px rgba(0,0,0,0.2);
  position: relative;
```

Experiment No. 11

Title: Demonstrate different color definitions in CSS (hex, RGB, names).

Aim:

To create a web page that demonstrates various methods of defining colors in CSS, including **color names**, **hexadecimal codes**, and **RGB values**.

Apparatus / Software Requirements:

- Computer with any OS
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

CSS (Cascading Style Sheets) allows defining colors for text, backgrounds, and borders using different formats:

1. **Named Colors** – Predefined color names recognized by browsers.
   - Example: red, blue, green, yellow.
2. **Hexadecimal Color Codes** – Colors defined using a 6-digit hexadecimal value representing red, green, and blue.
   - Format: #RRGGBB (values from 00 to FF).
   - Example: #FF0000 (red), #00FF00 (green), #0000FF (blue).
3. **RGB Color Values** – Colors defined using **RGB** function with decimal values ranging from 0 to 255.
   - Format: rgb(red, green, blue).
   - Example: rgb(255,0,0) (red).

Procedure:

1. Create a new HTML file and save it as **color-demo.html**.
2. Use HTML tags to create headings and paragraphs.
3. Apply CSS styles to demonstrate colors using **names**, **hex codes**, and **RGB values**.
4. Save the file and open it in a browser to view results.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Color Definitions</title>
  <style>
    /* Named Color */
    .named-color {
      color: red;
      background-color: lightyellow;
      padding: 10px;
    }

    /* Hexadecimal Color */
    .hex-color {
      color: #0066cc; /* Blue shade */
```

```
        background-color: #ffcccc; /* Light red shade */
        padding: 10px;
    }

    /* RGB Color */
    .rgb-color {
        color: rgb(0, 128, 0); /* Green */
        background-color: rgb(255, 255, 200); /* Light yellow */
        padding: 10px;
    }
  </style>
</head>
<body>

  <h1>Demonstrating Different CSS Color Definitions</h1>

  <p class="named-color">This text uses a named color (Red) for text and LightYellow for background.</p>
  <p class="hex-color">This text uses a hexadecimal color (#0066cc) for text and #ffcccc for background.</p>
  <p class="rgb-color">This text uses RGB color (0, 128, 0) for text and RGB(255, 255, 200) for background.</p>

</body>
</html>
```

Output:

When opened in a browser, the page will display three different paragraphs, each styled using a different CSS color definition method.

Result:

Successfully created a web page demonstrating **named colors**, **hexadecimal color codes**, and **RGB color values** in CSS.

Viva Questions:

1. What are the different ways to define colors in CSS?
2. What is the difference between hex code #FFFFFF and #FFF?
3. How many named colors are supported in CSS?
4. What are the advantages of using RGB over named colors?
5. Can we specify transparency with RGB values in CSS? If yes, how?

Experiment No. 12

Title: Use CSS to place a background image fixed at the center and repeat horizontally.

Aim:

To create a web page that uses CSS to set a background image fixed at the center of the page and repeated horizontally across the viewport.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Mozilla Firefox, Microsoft Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)
- Background image file (e.g., background.jpg)

Theory:

In CSS, the background properties allow developers to control how background images are displayed on web pages.
Key properties used:

1. background-image – Specifies the image to be used as background.
2. background-image: url('image.jpg');
3. background-repeat – Controls how the image repeats.
    - repeat-x: Repeats horizontally.
    - repeat-y: Repeats vertically.
    - no-repeat: No repetition.
4. background-position – Sets the starting position of the image.
    - Example: center center places the image at the center both vertically and horizontally.
5. background-attachment – Determines if the background scrolls or remains fixed.
    - fixed: Image stays in place when scrolling.

Procedure:

1. Choose or download a background image (background.jpg) and place it in the same folder as your HTML file.
2. Create a new HTML file and save it as background-demo.html.
3. Write HTML structure and CSS rules to set:
    - Image as background.
    - Image fixed at center.
    - Horizontal repetition only.
4. Save the file and open it in a browser to verify output.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Background Image - Fixed Center Repeat X</title>
  <style>
    body {
      background-image: url('background.jpg');
      background-repeat: repeat-x; /* Repeat horizontally */
```

```
            background-position: center center; /* Center alignment */
            background-attachment: fixed; /* Fixed position */
            margin: 0;
            height: 100vh;
        }

        .content {
            height: 2000px; /* To test scrolling */
            text-align: center;
            padding-top: 50px;
            font-family: Arial, sans-serif;
        }
    </style>
</head>
<body>

    <div class="content">
        <h1>Background Image Demo</h1>
        <p>Scroll down to see the background image fixed at the center and repeating horizontally.</p>
    </div>

</body>
</html>
```

Output:

When viewed in a browser:

- The background image stays fixed at the center position.
- The image repeats horizontally across the page width.
- When scrolling, the background does not move.

Result:

Successfully created a webpage that displays a background image fixed at the center with horizontal repetition using CSS properties.

Viva Questions:

1. What is the difference between background-repeat: repeat-x and background-repeat: repeat-y?
2. How do you prevent a background image from repeating?
3. What is the effect of background-attachment: fixed?
4. Can we use multiple background images in CSS? If yes, how?
5. What is the default value of background-repeat in CSS?

Experiment No. 13

Title: Style text with font size, weight, style, alignment, transformation, and decoration.

Aim:

To create a web page that demonstrates different text styling options in CSS, including **font size**, **font weight**, **font style**, **text alignment**, **text transformation**, and **text decoration**.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

CSS provides several properties to style and format text:

1. **font-size** – Sets the size of the text.
    - Example: font-size: 20px;
2. **font-weight** – Defines the thickness/boldness of the text.
    - Values: normal, bold, bolder, lighter, or numeric values (100–900).
3. **font-style** – Defines text style.
    - Values: normal, italic, oblique.
4. **text-align** – Aligns text horizontally within an element.
    - Values: left, right, center, justify.
5. **text-transform** – Controls the capitalization of text.
    - Values: none, capitalize, uppercase, lowercase.
6. **text-decoration** – Adds decoration to text.
    - Values: none, underline, overline, line-through.

Procedure:

1. Create a new HTML file and save it as **text-style-demo.html**.
2. Add headings and paragraphs to demonstrate different text styles.
3. Apply CSS rules to change font size, weight, style, alignment, transformation, and decoration.
4. Save and open the file in a browser to observe results.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Text Styling</title>
  <style>
    .font-size {
      font-size: 24px;
    }
    .font-weight {
      font-weight: bold;
    }
    .font-style {
      font-style: italic;
```

```
        }
        .text-align {
            text-align: center;
        }
        .text-transform {
            text-transform: uppercase;
        }
        .text-decoration {
            text-decoration: underline;
        }
    </style>
</head>
<body>

    <h1>Demonstrating CSS Text Styling</h1>

    <p class="font-size">This text has a font size of 24px.</p>
    <p class="font-weight">This text is bold using font-weight.</p>
    <p class="font-style">This text is italic using font-style.</p>
    <p class="text-align">This text is centered using text-align.</p>
    <p class="text-transform">This text is transformed to uppercase.</p>
    <p class="text-decoration">This text is underlined using text-decoration.</p>

</body>
</html>
```

Output:

When viewed in a browser:

- Each paragraph shows a specific text style change.
- Font size, boldness, italics, alignment, uppercase transformation, and underline effects are clearly visible.

Result:

Successfully created a webpage demonstrating **font size**, **font weight**, **font style**, **text alignment**, **text transformation**, and **text decoration** in CSS.

Viva Questions:

1. What is the difference between font-style: italic and font-style: oblique?
2. Can we combine text-transform and text-decoration?
3. What are relative units (em, %) in font size, and when are they useful?
4. How can you justify text using CSS?
5. What numeric values can be used for font-weight in CSS?

Experiment No. 14

Title: Illustrate the CSS Box Model using content, padding, border, and margin differences.

Aim:

To create a web page that demonstrates the CSS Box Model, highlighting differences between **content**, **padding**, **border**, and **margin**.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

The **CSS Box Model** is the foundation for layout design in CSS. It describes how every HTML element is represented as a rectangular box consisting of the following areas:

1. **Content** – The actual text, image, or other elements inside the box.
    - Controlled by properties like width and height.
2. **Padding** – Space between the content and the border.
    - Does not have color unless background color is applied to the element.
3. **Border** – A line surrounding the padding (and content).
    - Controlled using properties like border-width, border-style, and border-color.
4. **Margin** – Space between the element's border and neighboring elements.
    - Transparent and used for element spacing.

Box Model Formula:

Total Element Width = Content Width + Padding Left + Padding Right + Border Left + Border Right + Margin Left + Margin Right

Total Element Height = Content Height + Padding Top + Padding Bottom + Border Top + Border Bottom + Margin Top + Margin Bottom

Procedure:

1. Create a new HTML file and save it as **box-model-demo.html**.
2. Add a div element with some text content.
3. Apply CSS to set width, height, padding, border, and margin for the div.
4. Use different colors for content background, padding, and border to visualize differences.
5. Save the file and open it in a browser.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Box Model Demo</title>
  <style>
    .box {
      width: 200px; /* Content width */
      height: 100px; /* Content height */
      background-color: lightblue; /* Content area color */
```

```
            padding: 20px; /* Space inside border */
            border: 5px solid blue; /* Border style */
            margin: 30px; /* Space outside border */

            font-family: Arial, sans-serif;
            text-align: center;
        }

        body {
            background-color: #f0f0f0;
        }
    </style>
</head>
<body>

    <h1>CSS Box Model Illustration</h1>
    <div class="box">
        Content Area
    </div>

    <p>Observe: Content (light blue), Padding (inside border), Border (blue outline), Margin (grey space outside).</p>

</body>
</html>
```

Output:

When viewed in a browser:

- A **light blue box** will be displayed with text "Content Area".
- **Padding** increases the space between the text and the border.
- **Border** surrounds the padding area.
- **Margin** creates space between the box and surrounding elements.

Result:

Successfully created a webpage illustrating the CSS Box Model and differences between **content**, **padding**, **border**, and **margin**.

Viva Questions:

1. What are the four main components of the CSS Box Model?
2. How does padding differ from margin?
3. How is total element width and height calculated in CSS?
4. Can padding have a background color? Why or why not?
5. What is the difference between border-box and content-box in CSS box-sizing?

UNIT IV:

UNIT IV: JavaScript – Basics, I/O, Data Types and Logic Embedding JavaScript (Internal & External),Input and Output, Type Conversion, Conditional Statements & Loops, JavaScript Objects: Pre-defined and User-defined
Experiments:
1. Create web pages with both internal and external JavaScript.
2. Write programs to demonstrate different methods of input and output.
3. Create a voting eligibility checker using prompt, conditional logic, and table output.
4. Use window, document, math, array, string, date, and regex objects in different programs.
5. Validate registration form fields (Name, Mobile, Email) using JavaScript.
6. Perform logic checks using JavaScript: o Armstrong number o Denomination breakdown of an amount o Displaying weekdays using switch o Print 1 to 10 using different loops

UNIT IV: JavaScript – Basics, I/O, Data Types and Logic
(Embedding JavaScript, Input & Output, Type Conversion, Conditional Statements, Loops, JavaScript Objects)

4.1. Introduction to JavaScript

- JavaScript (JS) is a lightweight, interpreted programming language used to make web pages interactive.
- Runs in the browser, but also on servers (Node.js).
- Enhances **HTML (structure)** and **CSS (style)** with **logic and interactivity**.

4.2. Embedding JavaScript

a) Internal JavaScript
Placed inside the <script> tag in an HTML file.

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal JS</title>
</head>
<body>
  <h2>Internal JavaScript Example</h2>
  <script>
    alert("Hello! This is Internal JavaScript.");
  </script>
</body>
</html>
```

b) External JavaScript
Stored in a separate .js file and linked with <script src="...">.

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

script.js

```
alert("Hello from External JS!");
```

4.3. Input and Output in JavaScript

a) Output Methods

- alert("message") → Pop-up box.

- document.write("text") → Writes into the webpage.
- console.log("message") → Prints in browser console.
- innerHTML → Updates HTML content.

```
document.getElementById("demo").innerHTML = "Hello, JavaScript!";
```

b) Input

- prompt("Enter your name:") → Accepts user input.

```
let name = prompt("Enter your name:");
alert("Welcome " + name);
```

4.4. JavaScript Data Types

JavaScript is **dynamically typed** (no need to declare type explicitly).

a) Primitive Data Types

- **String** → "Hello", 'JS'
- **Number** → 10, 3.14
- **Boolean** → true, false
- **Undefined** → Variable declared but not assigned.
- **Null** → Empty or no value.
- **Symbol** (unique identifiers).
- **BigInt** → For very large integers.

b) Non-Primitive

- **Objects** (arrays, functions, user-defined objects).

```
let name = "Alice";     // String
let age = 21;           // Number
let isStudent = true;   // Boolean
let car = null;         // Null
let address;            // Undefined
```

4.5. Type Conversion

JavaScript automatically converts types (type coercion) or you can convert manually.

a) String Conversion

```
let num = 10;
let str = String(num); // "10"
```

b) Number Conversion

```
let str = "123";
let num = Number(str); // 123
```

c) Boolean Conversion

- Boolean(0) → false
- Boolean(1) → true
- Boolean("") → false
- Boolean("hello") → true

4.6. Conditional Statements

Used to make decisions in programs.

a) if Statement

```
let age = 18;
if (age >= 18) {
  console.log("Eligible to vote");
}
```

b) if...else

```
if (age < 18) {
  console.log("Minor");
} else {
  console.log("Adult");
}
```

c) if...else if...else

```
let marks = 75;
if (marks >= 90) console.log("Grade A");
else if (marks >= 75) console.log("Grade B");
else console.log("Grade C");
```

d) Switch Statement

```
let day = 2;
switch(day) {
  case 1: console.log("Monday"); break;
  case 2: console.log("Tuesday"); break;
  default: console.log("Other day");
}
```

4.7. Loops in JavaScript

a) for Loop

```
for (let i = 1; i <= 5; i++) {
  console.log("Number: " + i);
}
```

b) while Loop

```
let i = 1;
while (i <= 5) {
  console.log(i);
  i++;
}
```

c) do...while Loop

```
let i = 1;
do {
  console.log(i);
  i++;
} while (i <= 5);
```

d) for...in Loop (iterate over object properties)

```
let student = {name: "John", age: 20};
for (let key in student) {
  console.log(key + ": " + student[key]);
}
```

e) for...of Loop (iterate over iterable objects like arrays)

```
let arr = [10, 20, 30];
for (let val of arr) {
  console.log(val);
}
```

4.8. JavaScript Objects

Objects group related data and functions.

a) Predefined Objects

- **Math** → Provides mathematical functions.

```
console.log(Math.sqrt(16)); // 4
console.log(Math.random()); // Random number 0–1
```

- **Date** → Works with dates and times.

```
let now = new Date();
console.log(now.getFullYear());
```

- **String, Array, Boolean, Number** → Built-in wrappers with useful methods.

b) User-Defined Objects

Created using object literals, constructors, or classes.

```
let student = {
  name: "Alice",
  age: 20,
  greet: function() {
    console.log("Hello, " + this.name);
  }
};

student.greet(); // Hello, Alice
```

Experiment No.: 15

Title: Create web pages with both internal and external JavaScript.

Aim:

To develop web pages that demonstrate the use of **internal** and **external** JavaScript for adding dynamic behavior.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

JavaScript can be included in an HTML document in two main ways:

1. **Internal JavaScript** – Code is written inside <script> tags within the HTML file.
2. <script>
3. alert("Hello from internal JavaScript!");
4. </script>
5. **External JavaScript** – Code is stored in a separate .js file and linked to the HTML file using the src attribute of <script> tag.
6. <script src="script.js"></script>

Advantages of External JavaScript:

- Reusability across multiple pages.
- Cleaner HTML structure.
- Easier maintenance.

Procedure:

1. Create a new folder for the project.
2. Inside it, create:
    - An HTML file (internal-external-demo.html).
    - A JavaScript file (external.js).
3. Write internal JavaScript in the HTML file and external JavaScript in the .js file.
4. Link the external script using the <script> tag with src attribute.
5. Open the HTML file in a browser and observe the output.

Program Code:

internal-external-demo.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal and External JavaScript</title>
  <script>
    // Internal JavaScript
    function internalMessage() {
      alert("Hello! This is internal JavaScript.");
    }
  </script>
</head>
```

```
<body>

   <h1>JavaScript Demo</h1>
   <button onclick="internalMessage()">Run Internal JavaScript</button>
   <button onclick="externalMessage()">Run External JavaScript</button>

   <!-- Linking External JavaScript -->
   <script src="external.js"></script>
</body>
</html>
```

---

external.js

```
// External JavaScript
function externalMessage() {
   alert("Hello! This is external JavaScript.");
}
```

Output:

When the HTML page is opened in a browser:

- Clicking **"Run Internal JavaScript"** shows an alert message from internal JavaScript.
- Clicking **"Run External JavaScript"** shows an alert message from external JavaScript.

Result:

Successfully created web pages that demonstrate the usage of both internal and external JavaScript.

Viva Questions:

1. What is the difference between internal and external JavaScript?
2. Why is external JavaScript preferred for larger projects?
3. Can an HTML page use both internal and external JavaScript at the same time?
4. How do you link an external JavaScript file?
5. What happens if you place the <script> tag in the <head> versus at the end of <body>?

Experiment No.: 16

Title: Write programs to demonstrate different methods of input and output.

Aim:

To develop JavaScript programs that demonstrate various methods of **input** and **output** operations in a web page.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

In JavaScript, we can use multiple methods for taking **input** from the user and producing **output** on the screen.

Common Input Methods:

1. prompt() – Displays a dialog box that takes user input.
2. HTML <input> elements – Combined with JavaScript event handling.

Common Output Methods:

1. alert() – Displays a pop-up message box.
2. document.write() – Writes directly to the HTML document.
3. console.log() – Prints output to the browser console (for debugging).
4. DOM manipulation (e.g., innerHTML) – Displays output in a specific HTML element.

Procedure:

1. Create a new HTML file and save it as **io-methods-demo.html**.
2. Write JavaScript code to demonstrate:
    - Input using prompt() and HTML input field.
    - Output using alert(), document.write(), console.log(), and innerHTML.
3. Save the file and open it in a browser.
4. Test each method to observe the behavior.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Input and Output Methods</title>
  <script>
    // Using prompt() for input
    var name = prompt("Enter your name:");

    // Output using alert()
    alert("Hello, " + name + "! (alert output)");

    // Output using console.log()
    console.log("Hello, " + name + "! (console output)");
```

```
    // Output using document.write()
    document.write("<p>Hello, " + name + "! (document.write output)</p>");

    // Function to take input from HTML form and show output
    function showOutput() {
      var userInput = document.getElementById("userInput").value;
      document.getElementById("outputDiv").innerHTML =
        "You entered: " + userInput + " (innerHTML output)";
    }
  </script>
</head>
<body>

  <h2>HTML Input and Output Example</h2>
  <input type="text" id="userInput" placeholder="Type something here">
  <button onclick="showOutput()">Show Output</button>

  <div id="outputDiv" style="margin-top:10px; color:blue; font-weight:bold;"></div>

</body>
</html>
```

Output:

When viewed in a browser:

1. A prompt asks for your name.
2. An alert box displays a greeting.
3. The greeting is also logged in the console.
4. The greeting appears on the webpage using document.write().
5. When text is typed into the input field and the button is clicked, the output appears inside a element using innerHTML.

Result:

Successfully demonstrated **different methods of input and output** in JavaScript, including prompt(), alert(), console.log(), document.write(), and HTML input with innerHTML.

Viva Questions:

1. What is the difference between alert() and console.log() in JavaScript?
2. Why is document.write() not recommended for dynamic content after page load?
3. How do you take numeric input from a user using prompt()?
4. How is DOM manipulation different from using document.write()?
5. Can JavaScript read input from HTML forms without using prompt()? How?

Experiment No.: 17

Title: Create a voting eligibility checker using prompt, conditional logic, and table output.

Aim:

To write a JavaScript program that checks voting eligibility based on the user's age, using prompt() for input, conditional statements for decision-making, and a table for displaying output.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

Key Concepts Used:

1. **prompt()** – Used to get input from the user via a pop-up dialog.
2. var age = prompt("Enter your age:");
3. **Conditional Logic** – Decision-making using if, else if, else.
4. if (age >= 18) {
5. // eligible
6. } else {
7. // not eligible
8. }
9. **Table Output** – Using HTML table tags (<table>, <tr>, <td>) to display results clearly.
10. **Type Conversion** – Converting prompt() string input to a number using Number() to ensure accurate comparison.

Procedure:

1. Create a new HTML file and save it as **voting-checker.html**.
2. Use prompt() to accept name and age from the user.
3. Convert age to a numeric value.
4. Use if-else conditional logic to check voting eligibility (age >= 18).
5. Display the result inside a formatted HTML table.
6. Save and open the file in a browser to verify the result.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Voting Eligibility Checker</title>
  <script>
    // Taking input from the user
    var name = prompt("Enter your name:");
    var age = Number(prompt("Enter your age:"));

    // Checking eligibility
    var eligibility;
```

```
        if (age >= 18) {
            eligibility = "Eligible to Vote";
        } else {
            eligibility = "Not Eligible to Vote";
        }

        // Displaying result in a table
        document.write("<h2>Voting Eligibility Result</h2>");
        document.write("<table border='1' cellpadding='8'>");
        document.write("<tr><th>Name</th><th>Age</th><th>Status</th></tr>");
        document.write("<tr><td>" + name + "</td><td>" + age + "</td><td>" + eligibility +
"</td></tr>");
        document.write("</table>");
    </script>
</head>
<body>
</body>
</html>
```

Output:

When executed in a browser:

1. The user is prompted to enter their **name** and **age**.
2. Based on the age entered:
    - If **age ≥ 18** → Status shows *"Eligible to Vote"*.
    - If **age < 18** → Status shows *"Not Eligible to Vote"*.
3. The result is displayed in a neatly formatted table with columns: **Name**, **Age**, **Status**.

Result:

Successfully created a program that takes user input, applies conditional logic to check voting eligibility, and displays the result in a table format.

Viva Questions:

1. Why is type conversion necessary when using prompt() input for numeric comparisons?
2. What is the difference between == and === in JavaScript?
3. How can we modify the program to check for a minimum age of 21 instead?
4. Can we use switch instead of if-else for this program? Why or why not?
5. How can we style the table using CSS for better readability?

Experiment No.: 18

Title: Use window, document, math, array, string, date, and regex objects in different programs.

Aim:

To write JavaScript programs demonstrating the use of various **pre-defined objects**: window, document, Math, Array, String, Date, and RegExp.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

JavaScript provides several **built-in (pre-defined) objects** that simplify programming tasks:

1. **window Object**
   - Represents the browser window.
   - Methods: alert(), confirm(), prompt(), setTimeout().
2. **document Object**
   - Represents the web page loaded in the browser.
   - Methods: getElementById(), write(), querySelector().
3. **Math Object**
   - Provides mathematical constants and functions.
   - Examples: Math.PI, Math.sqrt(), Math.random().
4. **Array Object**
   - Represents ordered lists of items.
   - Methods: push(), pop(), sort(), length.
5. **String Object**
   - Represents text data.
   - Methods: toUpperCase(), substring(), replace().
6. **Date Object**
   - Represents dates and times.
   - Methods: getFullYear(), getMonth(), getDay().
7. **RegExp Object**
   - Represents regular expressions for pattern matching.
   - Methods: .test(), .exec().

Procedure:

1. Create a new HTML file and save it as **predefined-objects-demo.html**.
2. Write JavaScript code to demonstrate each object with a simple example.
3. Use different output methods (document.write(), alert(), console.log()).
4. Save the file and open it in a browser to verify the results.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Pre-defined Objects Demo</title>
```

```
  <script>
    // 1. window object
    window.alert("This is an alert using the window object!");

    // 2. document object
    document.write("<h2>Document Object Example</h2>");
    document.write("This text is written using document.write().<br>");

    // 3. Math object
    var randomNum = Math.floor(Math.random() * 100) + 1;
    document.write("<h3>Math Object</h3>");
    document.write("Random Number (1-100): " + randomNum + "<br>");
    document.write("Square root of 25: " + Math.sqrt(25) + "<br>");
    document.write("Value of PI: " + Math.PI + "<br>");

    // 4. Array object
    var fruits = ["Apple", "Banana", "Mango"];
    fruits.push("Orange");
    document.write("<h3>Array Object</h3>");
    document.write("Fruits: " + fruits.join(", ") + "<br>");
    document.write("First fruit: " + fruits[0] + "<br>");
    document.write("Total fruits: " + fruits.length + "<br>");

    // 5. String object
    var text = "JavaScript is Fun!";
    document.write("<h3>String Object</h3>");
    document.write("Uppercase: " + text.toUpperCase() + "<br>");
    document.write("Substring (0-10): " + text.substring(0, 10) + "<br>");
    document.write("Replace 'Fun' with 'Awesome': " + text.replace("Fun", "Awesome") +
"<br>");

    // 6. Date object
    var today = new Date();
    document.write("<h3>Date Object</h3>");
    document.write("Today's Date: " + today.toDateString() + "<br>");
    document.write("Year: " + today.getFullYear() + "<br>");
    document.write("Month: " + (today.getMonth() + 1) + "<br>"); // Month index starts
from 0

    // 7. RegExp object
    var pattern = /JavaScript/i;
    var result = pattern.test(text);
    document.write("<h3>RegExp Object</h3>");
    document.write("Does the text contain 'JavaScript'? " + result + "<br>");
  </script>
</head>
<body>
</body>
</html>
```

Output:

When opened in a browser:

- An alert box appears from the window object.
- The web page displays examples of **document**, **Math**, **Array**, **String**, **Date**, and **RegExp** objects.
- Each section clearly shows how the object is used and its output.

Result:

Successfully demonstrated the use of **window**, **document**, **Math**, **Array**, **String**, **Date**, and **RegExp** objects with working examples.

Viva Questions:

1. What is the difference between window and document objects in JavaScript?
2. How does Math.random() work, and how can we get a number between 1 and 10?
3. How do you create a regular expression in JavaScript?
4. What is the difference between toUpperCase() and toLowerCase() in the String object?
5. How can we format a date in JavaScript?

Experiment No.: 19

Title: Validate registration form fields (Name, Mobile, Email) using JavaScript.

Aim:

To create a registration form and validate the **Name**, **Mobile Number**, and **Email** fields using JavaScript before submission.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

Form Validation ensures that user inputs meet the required criteria before submission. This prevents incorrect or malicious data from being processed.

- **Name Validation**: Should contain only alphabets and spaces.
  - Regex: /^[A-Za-z]+$/
- **Mobile Number Validation**: Should contain exactly 10 digits.
  - Regex: /^[0-9]{10}$/
- **Email Validation**: Should follow standard email format (e.g., abc@example.com).
  - Regex: /^[^\s@]+@[^\s@]+\.[^\s@]+$/

Advantages of JavaScript Form Validation:

- Faster feedback to users.
- Reduced server load.
- Better user experience.

Procedure:

1. Create an HTML file and save it as **form-validation.html**.
2. Design a registration form with input fields for Name, Mobile, and Email.
3. Write JavaScript validation functions using **Regular Expressions**.
4. Display error messages if inputs are invalid.
5. Test the form by entering both correct and incorrect values.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Registration Form Validation</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 30px; }
    .error { color: red; font-size: 14px; }
    form { width: 300px; }
    label { display: block; margin-top: 10px; }
    input { width: 100%; padding: 5px; }
    button { margin-top: 15px; padding: 8px 15px; }
  </style>
  <script>
    function validateForm() {
```

```
        var name = document.getElementById("name").value.trim();
        var mobile = document.getElementById("mobile").value.trim();
        var email = document.getElementById("email").value.trim();

        var nameRegex = /^[A-Za-z ]+$/;
        var mobileRegex = /^[0-9]{10}$/;
        var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

        var valid = true;

        // Name validation
        if (!nameRegex.test(name)) {
            document.getElementById("nameError").innerHTML = "Name should contain only
letters and spaces.";
            valid = false;
        } else {
            document.getElementById("nameError").innerHTML = "";
        }

        // Mobile validation
        if (!mobileRegex.test(mobile)) {
            document.getElementById("mobileError").innerHTML = "Mobile must be exactly
10 digits.";
            valid = false;
        } else {
            document.getElementById("mobileError").innerHTML = "";
        }

        // Email validation
        if (!emailRegex.test(email)) {
            document.getElementById("emailError").innerHTML = "Enter a valid email
address.";
            valid = false;
        } else {
            document.getElementById("emailError").innerHTML = "";
        }

        return valid;
    }
  </script>
</head>
<body>
  <h2>Registration Form</h2>
  <form onsubmit="return validateForm()">
    <label>Name:</label>
    <input type="text" id="name" placeholder="Enter your name">
    <span id="nameError" class="error"></span>

    <label>Mobile:</label>
```

```
    <input type="text" id="mobile" placeholder="Enter 10-digit mobile">
    <span id="mobileError" class="error"></span>

    <label>Email:</label>
    <input type="text" id="email" placeholder="Enter your email">
    <span id="emailError" class="error"></span>

    <button type="submit">Register</button>
  </form>
</body>
</html>
```

Output:

- If **all fields** are valid → Form submits successfully.
- If **any field** is invalid → Error message appears below the respective field, preventing form submission.

Result:

Successfully created a registration form and implemented **JavaScript-based validation** for Name, Mobile, and Email fields using regular expressions.

Viva Questions:

1. What is the purpose of form validation in JavaScript?
2. How do you validate an email format using regex?
3. What is the difference between client-side and server-side validation?
4. Why do we use the return false statement in form validation?
5. What happens if JavaScript is disabled in the browser during form submission?

Experiment No.: 20

Title: Perform logic checks using JavaScript:

1. Armstrong Number
2. Denomination Breakdown of an Amount
3. Displaying Weekdays using Switch
4. Print 1 to 10 using Different Loops

Aim:

To implement JavaScript programs that demonstrate various logic checks and control flow constructs for problem solving.

Apparatus / Software Requirements:

- Computer with any operating system
- Web browser (Google Chrome, Firefox, Edge, etc.)
- Text editor (VS Code, Sublime Text, Notepad++, etc.)

Theory:

1. Armstrong Number:
An Armstrong number (also called a narcissistic number) of n digits is a number such that the sum of its digits raised to the power n is equal to the number itself.
Example: $153 = 1^3 + 5^3 + 3^3$
2. Denomination Breakdown:
Determines the number of currency notes/coins required for a given amount using available denominations.
3. Weekdays using Switch:
Demonstrates the switch statement for decision making based on a number (1–7) representing days.
4. Printing 1 to 10 using Loops:
Demonstrates **for**, **while**, and **do...while** loops for iterative tasks.

Procedure:

1. Create an HTML file named **logic-checks.html**.
2. For each problem, write a separate JavaScript function.
3. Use prompt() for input where applicable and document.write() or console.log() for output.
4. Test each program separately to ensure correct functionality.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Logic Checks</title>
</head>
<body>
  <h2>JavaScript Logic Checks</h2>
  <script>
    // 1) Armstrong Number Check
    let num = parseInt(prompt("Enter a number to check Armstrong:"));
```

```
let temp = num;
let sum = 0;
let digits = num.toString().length;
while (temp > 0) {
   let digit = temp % 10;
   sum += digit ** digits;
   temp = Math.floor(temp / 10);
}
if (sum === num) {
   document.write(num + " is an Armstrong number.<br>");
} else {
   document.write(num + " is not an Armstrong number.<br>");
}

// 2) Denomination Breakdown
let amount = parseInt(prompt("Enter amount for denomination breakdown:"));
let denominations = [2000, 500, 200, 100, 50, 20, 10, 5, 2, 1];
document.write("<h3>Denomination Breakdown</h3>");
denominations.forEach(function(den) {
   let count = Math.floor(amount / den);
   if (count > 0) {
      document.write(den + " x " + count + "<br>");
      amount %= den;
   }
});

// 3) Displaying Weekdays using Switch
let dayNum = parseInt(prompt("Enter day number (1-7):"));
switch(dayNum) {
   case 1: document.write("Monday<br>"); break;
   case 2: document.write("Tuesday<br>"); break;
   case 3: document.write("Wednesday<br>"); break;
   case 4: document.write("Thursday<br>"); break;
   case 5: document.write("Friday<br>"); break;
   case 6: document.write("Saturday<br>"); break;
   case 7: document.write("Sunday<br>"); break;
   default: document.write("Invalid day number<br>");
}

// 4) Printing 1 to 10 using different loops
document.write("<h3>For Loop:</h3>");
for (let i = 1; i <= 10; i++) {
   document.write(i + " ");
}

document.write("<h3><br>While Loop:</h3>");
let i = 1;
while (i <= 10) {
   document.write(i + " ");
```

```
        i++;
      }

      document.write("<h3><br>Do...While Loop:</h3>");
      i = 1;
      do {
        document.write(i + " ");
        i++;
      } while (i <= 10);
    </script>
</body>
</html>
```

Output:

1. **Armstrong Number** → Displays whether the entered number is Armstrong or not.
2. **Denomination Breakdown** → Shows currency notes/coins required for the entered amount.
3. **Weekdays using Switch** → Displays the corresponding weekday for a given number.
4. **Printing 1 to 10** → Displays numbers 1 to 10 using for, while, and do...while loops.

Result:

Successfully implemented JavaScript programs to perform:

- Armstrong number check
- Denomination breakdown
- Weekday display using switch
- Loop-based number printing (for, while, do...while)

Viva Questions:

1. What is an Armstrong number? Give an example.
2. How does the switch statement differ from if-else?
3. Which loop guarantees at least one execution?
4. How can you optimize the denomination breakdown program?
5. Why do we use Math.floor() in these programs?

UNIT V:

UNIT V: JavaScript Functions, Events & Database Connectivity JavaScript Functions and Event Handling, Functional Programming Examples, DOM Manipulation, Intro to Server-side JavaScript with Node.js, Connecting to Databases (MySQL, MongoDB)

Experiments:

1. Create reusable JavaScript functions for: o Factorial o Fibonacci o Prime numbers o Palindrome check
2. Design a single-page application with text input and buttons for each of the above functions
using event handlers.
3. Use for-in, for-of, and for Each loops to print object data.
4. Introduction to Node.js environment – Setup and sample console.log() app.
5. Connect a sample JavaScript app to MySQL or MongoDB and perform basic CRUD operations
(optional/demonstration-based if not in lab scope).

UNIT V: JavaScript Functions, Events & Database Connectivity

(Functions, Event Handling, Functional Programming, DOM Manipulation, Node.js, Database Connectivity with MySQL & MongoDB)

5.1. JavaScript Functions

a) What is a Function?

- A **function** is a block of code designed to perform a particular task.
- Reusable → Write once, use many times.

b) Declaring a Function

```
function greet(name) {
  return "Hello, " + name;
}

console.log(greet("Alice")); // Output: Hello, Alice
```

c) Function Expressions

```
const add = function(a, b) {
  return a + b;
};
console.log(add(5, 3)); // 8
```

d) Arrow Functions (ES6)

```
const square = (x) => x * x;
console.log(square(4)); // 16
```

e) Default Parameters

```
function multiply(a, b = 2) {
  return a * b;
}
console.log(multiply(5)); // 10
```

5.2. Event Handling in JavaScript

Events are actions performed by the user (click, keypress, mouseover, etc.).

a) Inline Event

```
<button onclick="alert('Button clicked!')">Click Me</button>
```

b) DOM Event Handling

```
<button id="btn">Click</button>
<script>
  document.getElementById("btn").onclick = function() {
    alert("Button was clicked!");
  };
</script>
```

c) addEventListener()

```
document.getElementById("btn").addEventListener("click", function() {
  alert("Handled with addEventListener");
});
```

5.3. Functional Programming Examples

Functional programming treats functions as **first-class citizens**.

a) Higher-Order Functions

A function that accepts another function as parameter or returns one.

```
function applyOperation(x, y, operation) {
  return operation(x, y);
}

console.log(applyOperation(5, 3, (a, b) => a + b)); // 8
```

b) Array Functional Methods

- map() → Transform each element.
- filter() → Select elements.
- reduce() → Accumulate values.

```
let numbers = [1, 2, 3, 4, 5];
let squares = numbers.map(n => n * n);
let evens = numbers.filter(n => n % 2 === 0);
let sum = numbers.reduce((a, b) => a + b, 0);

console.log(squares); // [1,4,9,16,25]
console.log(evens);   // [2,4]
console.log(sum);     // 15
```

5.4. DOM Manipulation

The **DOM (Document Object Model)** represents HTML as a tree structure. JavaScript can dynamically modify it.

a) Selecting Elements

```
document.getElementById("id");
document.getElementsByClassName("class");
document.querySelector("div");
```

b) Changing Content & Style

```
document.getElementById("demo").innerHTML = "New Content";
document.getElementById("demo").style.color = "blue";
```

c) Creating and Appending Elements

```
let para = document.createElement("p");
para.innerText = "This is a new paragraph.";
```

```
document.body.appendChild(para);
```

5.5. Intro to Server-Side JavaScript with Node.js

a) What is Node.js?

- Open-source, server-side runtime environment for JavaScript.
- Built on **Google Chrome's V8 engine**.
- Allows JS to run outside the browser.

b) Creating a Simple Server

```
const http = require("http");

http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello from Node.js server');
}).listen(3000);

console.log("Server running at http://localhost:3000/");
```

5.6. Connecting to Databases

a) Connecting Node.js with MySQL

1. Install MySQL package:
2. npm install mysql
3. Example Code:

```
const mysql = require('mysql');

const con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "testdb"
});

con.connect(err => {
  if (err) throw err;
  console.log("Connected to MySQL!");

  con.query("SELECT * FROM students", (err, result) => {
    if (err) throw err;
    console.log(result);
  });
});
```

b) Connecting Node.js with MongoDB

1. Install MongoDB driver:
2. npm install mongodb
3. Example Code:

```
const { MongoClient } = require('mongodb');
const url = "mongodb://localhost:27017";
const client = new MongoClient(url);
```

```
async function run() {
  try {
    await client.connect();
    console.log("Connected to MongoDB");

    const db = client.db("school");
    const students = db.collection("students");

    await students.insertOne({ name: "Alice", age: 21 });
    const data = await students.find().toArray();
    console.log(data);
  } finally {
    await client.close();
  }
}
run().catch(console.dir);
```

Experiment No.: 21

Title: Create reusable JavaScript functions for:

1. Factorial
2. Fibonacci
3. Prime Numbers
4. Palindrome Check

Aim:

To write and execute reusable JavaScript functions for performing mathematical and logical operations like factorial, Fibonacci series, prime number checking, and palindrome checking.

Software / Hardware Requirements:

- Any operating system (Windows/Linux/Mac)
- Web browser (Chrome, Firefox, Edge)
- Text Editor (VS Code, Notepad++, Sublime Text)

Theory:

Functions in JavaScript:

A function is a block of reusable code that performs a particular task. Functions improve modularity, reduce redundancy, and make programs easy to maintain.

- **Factorial:** Product of all positive integers up to n. Example: 5! = 5 × 4 × 3 × 2 × 1 = 120.
- **Fibonacci Series:** A sequence where each number is the sum of the two preceding numbers. Example: 0, 1, 1, 2, 3, 5, ...
- **Prime Numbers:** Numbers greater than 1 that have no divisors other than 1 and itself. Example: 2, 3, 5, 7, 11.
- **Palindrome:** A word, number, or phrase that reads the same backward as forward. Example: 121, madam.

Procedure:

1. Create an HTML file and link JavaScript.
2. Define four separate reusable functions for factorial, Fibonacci, prime check, and palindrome.
3. Use user input (prompt()) to test each function.
4. Display results using document.write() or console output.
5. Verify correctness with different inputs.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Reusable JavaScript Functions</title>
</head>
<body>
  <h2>JavaScript Functions Example</h2>
  <script>
    // 1) Factorial Function
    function factorial(n) {
```

```
        let fact = 1;
        for (let i = 1; i <= n; i++) {
            fact *= i;
        }
        return fact;
    }

    // 2) Fibonacci Function
    function fibonacci(limit) {
        let series = [0, 1];
        for (let i = 2; i < limit; i++) {
            series[i] = series[i - 1] + series[i - 2];
        }
        return series;
    }

    // 3) Prime Number Function
    function isPrime(num) {
        if (num < 2) return false;
        for (let i = 2; i <= Math.sqrt(num); i++) {
            if (num % i === 0) return false;
        }
        return true;
    }

    // 4) Palindrome Function
    function isPalindrome(str) {
        str = str.toString().toLowerCase();
        let reversed = str.split("").reverse().join("");
        return str === reversed;
    }

    // Execution
    let num1 = parseInt(prompt("Enter a number for factorial:"));
    document.write("Factorial of " + num1 + " = " + factorial(num1) + "<br>");

    let fibCount = parseInt(prompt("Enter number of terms for Fibonacci:"));
    document.write("Fibonacci Series: " + fibonacci(fibCount).join(", ") + "<br>");

    let primeCheck = parseInt(prompt("Enter a number to check Prime:"));
    document.write(primeCheck + (isPrime(primeCheck) ? " is Prime<br>" : " is not
Prime<br>"));

    let palInput = prompt("Enter a string or number to check Palindrome:");
    document.write(palInput + (isPalindrome(palInput) ? " is a Palindrome" : " is not a
Palindrome"));
  </script>
</body>
</html>
```

Output:

1. Factorial of a given number is displayed.
2. Fibonacci sequence is generated up to the entered number of terms.
3. Input number is checked for primality.
4. String/number is checked for palindrome property.

Result:

Successfully created reusable JavaScript functions to calculate factorial, generate Fibonacci sequence, check for prime numbers, and verify palindrome strings/numbers.

Viva Questions:

1. What is the difference between a function declaration and a function expression in JavaScript?
2. How can recursion be used to calculate factorial or Fibonacci?
3. Why is Math.sqrt() used in prime number checking?
4. What are the advantages of writing reusable functions?
5. Can a palindrome function work for both numbers and strings? How?

Experiment No.: 22

Title: Design a single-page application with text input and buttons for each of the above functions using event handlers.

Aim:

To design and implement a single-page application (SPA) in JavaScript that takes user input through a text box and uses buttons with event handlers to perform different operations:

1. Factorial
2. Fibonacci
3. Prime Number Check
4. Palindrome Check

Software / Hardware Requirements:

- Any operating system (Windows/Linux/Mac)
- Web browser (Chrome/Firefox/Edge)
- Text Editor (VS Code / Sublime / Notepad++)

Theory:

- **Single Page Application (SPA):** A web application that dynamically updates content on a single HTML page without reloading.
- **Event Handling in JavaScript:** Events are user actions like clicks, input, or keypress. Event handlers (e.g., onclick, onchange) are used to trigger functions when events occur.
- **Functions used:**
  - **Factorial:** Calculates product of numbers up to n.
  - **Fibonacci:** Generates sequence where each number is sum of previous two.
  - **Prime Check:** Determines if a number is divisible only by 1 and itself.
  - **Palindrome:** Checks if a string/number reads the same forward and backward.

Procedure:

1. Create a single HTML page with a text input field.
2. Add four buttons for **Factorial, Fibonacci, Prime Check, and Palindrome**.
3. Define JavaScript functions for each operation.
4. Attach **event handlers** (onclick) to buttons.
5. Display the results dynamically on the same page without reloading.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Single Page Application - Functions</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    input, button { padding: 8px; margin: 5px; }
    button { border-radius: 6px; cursor: pointer; }
    #result { margin-top: 20px; font-weight: bold; color: darkblue; }
  </style>
</head>
```

```
<body>
  <h2>JavaScript Functions with Event Handlers</h2>
  <label>Enter a value: </label>
  <input type="text" id="userInput">
  <br>

  <button onclick="checkFactorial()">Factorial</button>
  <button onclick="checkFibonacci()">Fibonacci</button>
  <button onclick="checkPrime()">Prime Check</button>
  <button onclick="checkPalindrome()">Palindrome</button>

  <div id="result"></div>

  <script>
    // Factorial Function
    function factorial(n) {
      let fact = 1;
      for (let i = 1; i <= n; i++) fact *= i;
      return fact;
    }

    function checkFactorial() {
      let num = parseInt(document.getElementById("userInput").value);
      document.getElementById("result").innerHTML =
        "Factorial of " + num + " = " + factorial(num);
    }

    // Fibonacci Function
    function fibonacci(limit) {
      let series = [0, 1];
      for (let i = 2; i < limit; i++) {
        series[i] = series[i - 1] + series[i - 2];
      }
      return series;
    }

    function checkFibonacci() {
      let num = parseInt(document.getElementById("userInput").value);
      document.getElementById("result").innerHTML =
        "Fibonacci Series (" + num + " terms): " + fibonacci(num).join(", ");
    }

    // Prime Number Function
    function isPrime(num) {
      if (num < 2) return false;
      for (let i = 2; i <= Math.sqrt(num); i++) {
        if (num % i === 0) return false;
      }
      return true;
```

```
    }

    function checkPrime() {
      let num = parseInt(document.getElementById("userInput").value);
      document.getElementById("result").innerHTML =
        num + (isPrime(num) ? " is a Prime number." : " is not a Prime number.");
    }

    // Palindrome Function
    function isPalindrome(str) {
      str = str.toString().toLowerCase();
      return str === str.split("").reverse().join("");
    }

    function checkPalindrome() {
      let val = document.getElementById("userInput").value;
      document.getElementById("result").innerHTML =
        val + (isPalindrome(val) ? " is a Palindrome." : " is not a Palindrome.");
    }
  </script>
</body>
</html>
```

Output:

- User enters a value in the text box.
- Clicking **Factorial** displays factorial of the number.
- Clicking **Fibonacci** shows Fibonacci series up to entered terms.
- Clicking **Prime Check** tells whether the number is prime.
- Clicking **Palindrome** checks if input is palindrome.

Result:

Successfully designed a single-page application using JavaScript functions and event handlers for factorial, Fibonacci, prime number, and palindrome checking.

Viva Questions:

1. What is the difference between inline, internal, and external event handling in JavaScript?
2. Why are event handlers important in interactive web applications?
3. How does DOM manipulation help in building SPAs?
4. Can we reuse the same function across multiple event handlers?
5. What are advantages of using event listeners (addEventListener) over inline events (onclick)?

Experiment No.: 23

Title: Use for-in, for-of, and forEach loops to print object data.

Aim:

To write JavaScript programs that demonstrate different looping mechanisms (for-in, for-of, forEach) for iterating over arrays and objects.

Software / Hardware Requirements:

- Any operating system (Windows/Linux/Mac)
- Web browser (Chrome/Firefox/Edge)
- Text Editor (VS Code / Sublime / Notepad++)

Theory:

- **for-in loop:** Iterates over the **keys/properties** of an object.
- **for-of loop:** Iterates over **iterable objects** (like arrays, strings, maps).
- **forEach() method:** A **higher-order function** that executes a callback function for each element of an array.

Key Differences:

- for-in → works with object properties.
- for-of → works with values of iterable objects.
- forEach() → method on arrays, provides both element and index.

Procedure:

1. Define a JavaScript object and an array.
2. Use for-in to loop through object properties.
3. Use for-of to loop through array values.
4. Use forEach() to loop through array values and indexes.
5. Display output on the web page or console.

Program Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Loop Demonstration</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    pre { background: #f4f4f4; padding: 10px; border-radius: 6px; }
  </style>
</head>
<body>
  <h2>JavaScript Loops: for-in, for-of, forEach</h2>
  <div id="output"></div>
  <script>
    let student = {
      name: "Ravi Kumar",
      age: 21,
      course: "B.Tech",
      college: "MITS"
```

```
    };
    let subjects = ["HTML", "CSS", "JavaScript", "Node.js"];
    let result = "";                                          // Using for-in with object
    result += "<h3>Using for-in (Object properties):</h3><pre>";
    for (let key in student) {
      result += key + ": " + student[key] + "\n";
    }
    result += "</pre>";                                       // Using for-of with array
    result += "<h3>Using for-of (Array values):</h3><pre>";
    for (let subject of subjects) {
      result += subject + "\n";
    }
    result += "</pre>";                                       // Using forEach with array
    result += "<h3>Using forEach (Array with index):</h3><pre>";
    subjects.forEach((sub, index) => {
      result += index + " → " + sub + "\n";
    });
    result += "</pre>";
    document.getElementById("output").innerHTML = result;
  </script>
</body>
</html>
```

Sample Output (Displayed on Web Page):

Using for-in (Object properties):
name: Durga charan
age: 21
course: B.Tech
college: MITS
Using for-of (Array values):
HTML
CSS
JavaScript
Node.js
Using forEach (Array with index):
0 → HTML
1 → CSS
2 → JavaScript
3 → Node.js

Result:

Successfully demonstrated the usage of for-in, for-of, and forEach loops in JavaScript to iterate through object data and arrays.

Viva Questions:

1. What is the difference between for-in and for-of loops?
2. Can for-of be used directly with objects? Why or why not?
3. How does forEach() differ from a traditional loop?

4. Which loop is best suited for iterating object properties?
5. Can we break out of a forEach() loop? If not, what is the alternative?

Experiment No.: 24

Title: Introduction to Node.js environment – Setup and sample console.log() app

---

Aim:

To set up the Node.js runtime environment and run a simple program that uses console.log() to display output.

Software / Hardware Requirements:

- Operating System: Windows/Linux/Mac
- Node.js (latest stable version) installed from https://nodejs.org
- Text Editor: VS Code / Sublime / Notepad++
- Command Line / Terminal

Theory:

- **What is Node.js?**
  Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside the web browser.
  - Built on Chrome's **V8 JavaScript engine**.
  - Used for **server-side programming**.
  - Enables building **scalable, fast, event-driven applications**.
- **Features of Node.js:**

1. Asynchronous and Event-Driven.
2. Very Fast Execution.
3. Single-Threaded but Highly Scalable.
4. NPM (Node Package Manager) for libraries.
5. Cross-Platform support.

- **console.log():**
  A function in Node.js (and JavaScript) used to print messages or variable values to the console/terminal.

Procedure:

1. Download and install **Node.js** from https://nodejs.org.
2. Verify installation:
   - Open **Command Prompt / Terminal**.
   - Run:
   - node -v

(Displays Node.js version)

   - Run:
   - npm -v

(Displays npm version)

3. Create a file named app.js.
4. Write a simple program using console.log().
5. Execute the program in terminal using:
6. node app.js
7. Observe the output in the console.

Program Code (app.js):

```
// app.js
```

```
// Simple Node.js program

console.log("Welcome to Node.js!");
console.log("This is your first Node.js application.");
console.log("Node.js Version Check:");
console.log("Node.js is running successfully 🚀 ");
```

Sample Output (Terminal):

```
Welcome to Node.js!
This is your first Node.js application.
Node.js Version Check:
Node.js is running successfully 🚀
```

Result:

Successfully installed Node.js environment and executed a simple console-based program using console.log().

Viva Questions:

1. What is Node.js and how is it different from JavaScript in the browser?
2. What engine powers Node.js?
3. How do you check the installed Node.js version?
4. What is the purpose of console.log()?
5. What is npm in Node.js?

Experiment No.: 25

Title: Connect a sample JavaScript app to MySQL or MongoDB and perform basic CRUD operations

---

Aim:

To demonstrate how to connect a JavaScript (Node.js) application to a database (**MySQL or MongoDB**) and perform basic CRUD (Create, Read, Update, Delete) operations.

Software / Hardware Requirements:

- Operating System: Windows/Linux/Mac
- **Node.js** installed (https://nodejs.org)
- **Database:**
  - MySQL (via mysql2 npm package) OR
  - MongoDB (via mongodb npm package / MongoDB Compass)
- Text Editor: VS Code
- Command Line / Terminal

Theory:

- **CRUD Operations:**
  CRUD stands for **Create, Read, Update, Delete** – the four basic operations used to manage data in a database.
- **MySQL & MongoDB:**
  - **MySQL:** A relational database using structured tables (SQL).
  - **MongoDB:** A NoSQL database using documents in JSON-like format.
- **Node.js Database Connectivity:**
  - Install drivers/libraries using npm.
  - Connect using a **connection string**.
  - Perform queries using JavaScript functions.

Procedure:

1. Install Node.js and verify with node -v.
2. Create a project folder (e.g., db-demo).
3. Initialize Node.js project:
4. npm init -y
5. Install database driver:
   - For MySQL: npm install mysql2
   - For MongoDB: npm install mongodb
6. Write JavaScript code to connect to the database.
7. Implement basic CRUD operations.
8. Run the app with:
9. node app.js
10. Verify results in terminal / database.

Program Codes:

Option A: MySQL Example (app.js)

```
// Import MySQL module
const mysql = require('mysql2');
```

```
// Create connection
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',  // replace with your MySQL password
  database: 'studentdb'
});

// Connect to MySQL
connection.connect(err => {
  if (err) throw err;
  console.log("Connected to MySQL!");

  // CREATE (Insert Data)
  connection.query("INSERT INTO students (id, name, age) VALUES (1, 'Alice', 20)", (err)
=> {
    if (err) throw err;
    console.log("Record Inserted!");
  });

  // READ (Select Data)
  connection.query("SELECT * FROM students", (err, results) => {
    if (err) throw err;
    console.log("Data Retrieved:", results);
  });

  // UPDATE
  connection.query("UPDATE students SET age=21 WHERE id=1", (err) => {
    if (err) throw err;
    console.log("Record Updated!");
  });

  // DELETE
  connection.query("DELETE FROM students WHERE id=1", (err) => {
    if (err) throw err;
    console.log("Record Deleted!");
    connection.end();
  });
});
```

Option B: MongoDB Example (app.js)

```
// Import MongoDB client
const { MongoClient } = require('mongodb');

// Connection URL
const url = 'mongodb://127.0.0.1:27017';
const client = new MongoClient(url);

// Database and Collection
```

```
const dbName = 'studentdb';
const collectionName = 'students';

async function run() {
  try {
    await client.connect();
    console.log("Connected to MongoDB!");

    const db = client.db(dbName);
    const collection = db.collection(collectionName);

    // CREATE (Insert)
    await collection.insertOne({ name: "Alice", age: 20 });
    console.log("Record Inserted!");
    // READ (Find)
    const students = await collection.find().toArray();
    console.log("Data Retrieved:", students);
    // UPDATE
    await collection.updateOne({ name: "Alice" }, { $set: { age: 21 } });
    console.log("Record Updated!");
    // DELETE
    await collection.deleteOne({ name: "Alice" });
    console.log("Record Deleted!");
  } finally {
    await client.close();
  }
}
run().catch(console.dir);
```

Sample Output (MongoDB Example):

```
Connected to MongoDB!
Record Inserted!
Data Retrieved: [ { _id: ObjectId("66b54f..."), name: 'Alice', age: 20 } ]
Record Updated!
Record Deleted!
```

Result:

Successfully connected a JavaScript app to a database (**MySQL/MongoDB**) and performed basic CRUD operations.

Viva Questions:

1. What is the difference between MySQL and MongoDB?
2. What does CRUD stand for?
3. How do you install database drivers in Node.js?
4. Why is MongoDB called NoSQL?
5. What are advantages of using Node.js for database connectivity?