Rayat Shikshan Sanstha's

Karmaveer Bhaurao Patil College of Engineering, Satara

Computer Science & Engineering Department

Seminar II

(BTCOS407)

Class: Third Year

Computer Engineering

SEMESTER VI

Academic year

2024-25

Experiment 1: Designing an Interactive HTML Form

Title:

Design an HTML form for displaying information using interactive CSS, including images and tables.

Objective:

To create an HTML form with interactive CSS, including images and tables, to enhance the user interface and improve data presentation.

Theory:

HTML forms allow users to enter and submit data, which can be processed using backend technologies or JavaScript. CSS enhances the appearance and interactivity of forms by applying styles, animations, and layout structures. Tables are useful for organizing and displaying data systematically. Images add visual appeal and branding.

Explanation of HTML Tags Used:

`<!DOCTYPE html>`

Declares the document type and version of HTML (HTML5 in this case).

`<html lang="en">`

The root element of the document. The `lang="en"` attribute specifies that the document is in English.

`<head>`

Contains metadata, styles, and other non-visible elements.

`<meta charset="UTF-8">`

Specifies the character encoding for the document (UTF-8 supports all characters and symbols).

`<meta name="viewport" content="width=device-width, initial-scale=1.0">`

Ensures responsive behavior on different screen sizes.

`<title>Interactive Information Form</title>`

Sets the title of the webpage (visible in the browser tab).

`<style>`

Contains internal CSS for styling the webpage.

`<body>`

- The main content of the webpage.

`<div class="container">`

1. A division (`<div>`) used to wrap all elements inside a container for styling purposes.

`<h2>`

1. A heading tag (level 2) used for section titles.

`<form>`

1. Defines an HTML form where users can input data.

`<label>`

- A text label for input fields, making the form more accessible.

`<input type="text">`

- A text input field for the user to enter their name.
  - `type="text"`: Specifies that it's a text field.
  - `id="name"` and `name="name"`: Used for identifying the input field.
  - `required`: Ensures the field cannot be left empty.

`<input type="email">`

An email input field that only accepts valid email formats.

`<select>` and `<option>`

A dropdown menu for selecting gender.

`<select>`: The main dropdown container.

`<option>`: Each selectable item inside the dropdown.

`<button type="submit">Submit</button>`

A button that submits the form when clicked.

`<table>`

Defines a table for displaying structured information.

`<tr>`

- Defines a table row.

`<th>`

1. Defines a table header cell (bold and centered by default).

`<td>`

- Defines a table data cell.

`<div class="image-container">`

1. A wrapper for the image section.

`<img src="https://via.placeholder.com/100" alt="Profile">`

1. Displays an image.
    1. `src="..."`: Specifies the image URL.
    2. `alt="Profile"`: Alternate text for accessibility.

---

2. CSS Properties Explained

Body Styling

```
css
CopyEdit
body {
    font-family: Arial, sans-serif;
    margin: 20px;
    background-color: #f4f4f4;
}
```

- Sets font style, spacing, and background color.

Container Styling

```
css
CopyEdit
.container {
    width: 50%;
    margin: auto;
    background: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
}
```

Centers the content and gives it a box-shadow for a modern look.

Button Styling

```
css
CopyEdit
button {
    width: 100%;
    padding: 10px;
    background: #28a745;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background 0.3s;
}
button:hover {
    background: #218838;
}
```

Styles the button with a green color and adds a hover effect.

Table Styling

```
css
CopyEdit
.info-table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
}
.info-table th, .info-table td {
    border: 1px solid #ddd;
    padding: 10px;
    text-align: left;
}
.info-table th {
    background: #007bff;
    color: white;
}
.info-table tr:hover {
    background: #f1f1f1;
}
```

Adds styling to the table, with colored headers and hover effects.

Image Styling

```
css
CopyEdit
.image-container img {
    width: 100px;
    height: auto;
    border-radius: 50%;
    transition: transform 0.3s;
}
.image-container img:hover {
    transform: scale(1.1);
}
```

Makes the image circular and adds a zoom effect on hover.

Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Interactive Information Form</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            background-color: #f4f4f4;
        }
        .container {
            width: 50%;
            margin: auto;
            background: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
        }
        h2 {
            text-align: center;
            color: #333;
        }
        label {
            font-weight: bold;
        }
        input, select {
            width: 100%;
            padding: 8px;
            margin: 5px 0;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        button {
            width: 100%;
            padding: 10px;
            background: #28a745;
            color: white;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            transition: background 0.3s;
        }
        button:hover {
            background: #218838;
        }
        .info-table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 20px;
        }
        .info-table th, .info-table td {
            border: 1px solid #ddd;
            padding: 10px;
            text-align: left;
        }
        .info-table th {
            background: #007bff;
            color: white;
        }
        .info-table tr:hover {
            background: #f1f1f1;
        }
        .image-container {
            text-align: center;
            margin-top: 20px;
        }
        .image-container img {
            width: 100px;
            height: auto;
            border-radius: 50%;
            transition: transform 0.3s;
        }
        .image-container img:hover {
            transform: scale(1.1);
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>User Information Form</h2>
        <form>
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>

            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>

            <label for="gender">Gender:</label>
            <select id="gender" name="gender">
                <option value="male">Male</option>
                <option value="female">Female</option>
                <option value="other">Other</option>
            </select>

            <button type="submit">Submit</button>
        </form>

        <h2>Information Table</h2>
        <table class="info-table">
            <tr>
                <th>Name</th>
                <th>Email</th>
                <th>Gender</th>
            </tr>
            <tr>
                <td>John Doe</td>
                <td>john.doe@example.com</td>
                <td>Male</td>
            </tr>
            <tr>
                <td>Jane Smith</td>
                <td>jane.smith@example.com</td>
                <td>Female</td>
            </tr>
        </table>

        <div class="image-container">
            <h2>Profile Image</h2>
            <img src="https://via.placeholder.com/100" alt="Profile">
        </div>
    </div>
</body>
</html>
```

Conclusion:

This experiment demonstrated how to create an interactive HTML form with CSS styling, images, and tables, improving the presentation and usability of the webpage.

Experiment 2: Department Webpage

Title:

Create a webpage describing your department with background color, navigation links, images, and font styling.

Objective:

To develop a well-structured HTML webpage for the department, incorporating background colors, navigation links, images, font styling, and tables for better organization and presentation.

Theory:

A well-structured webpage is essential for presenting information clearly and interactively. HTML provides the structure, while CSS enhances the appearance with styling features such as background colors, fonts, and interactive elements. Navigation links allow easy movement within the page.

Explanation of HTML Tags Used:

1. Basic Document Structure

`<!DOCTYPE html>`

- Declares that the document follows the **HTML5** standard.

`<html lang="en">`

- The root element of the document.
- The `lang="en"` attribute specifies that the language of the content is English.

`<head>`

1. Contains metadata and styling information.

`<meta charset="UTF-8">`

1. Defines the character encoding for the webpage (**UTF-8**, which supports various characters and symbols).

`<meta name="viewport" content="width=device-width, initial-scale=1.0">`

1. Ensures the webpage is **responsive** by making it adapt to different screen sizes.

`<title>Department Page</title>`

- Sets the **title of the webpage** (visible in the browser tab).

2. Styling with CSS

`<style>`

Contains **internal CSS** to style the webpage elements.

3. Body Section (Visible Content)

`<body>`

Contains the main content of the webpage.

4. Header Section

`<header>`

A semantic tag that defines the **top section** of the webpage.

Used here to **display the department name**.

```
<header>
    Welcome to the Department of Computer Science
</header>
```

5. Navigation Menu

`<nav>`

A semantic tag that represents the **navigation bar**.

`<a href="#section">`

- Creates **navigation links** that allow users to jump to different sections within the same page.

```
<nav>
    <a href="#about">About</a>
    <a href="#faculty">Faculty</a>
    <a href="#research">Research</a>
    <a href="#contact">Contact</a>
</nav>
```

1. **`href="#about"`**: Links to the **section** with `id="about"`.

6. Main Content Sections

`<div class="container" id="about">`

- The `<div>` tag is used as a **container** for organizing content.
- **`class="container"`** applies CSS styles to maintain structure and spacing.
- **`id="about"`** makes this section identifiable for navigation.

```
<div class="container" id="about">
    <h2>About Our Department</h2>
    <p>The Department of Computer Science is dedicated to excellence in
teaching, research, and innovation.</p>
</div>
```

1. **`<h2>`**: Defines a **section heading**.
2. **`<p>`**: Defines a **paragraph** for content.

The same structure is used for **faculty, research, and contact** sections.

7. Image Section

`<div class="container image-container">`

- A `<div>` that groups the **department image** and applies styling.

`<img src="https://via.placeholder.com/300" alt="Department Image">`

- **`<img>`** is an image tag.
- **`src="..."`**: Specifies the image URL.
- **`alt="..."`**: Alternative text (shown if the image does not load).

```
<div class="container image-container">
    <h2>Department Image</h2>
    <img src="https://via.placeholder.com/300" alt="Department Image">
</div>
```

8. Footer Section

`<footer>`

A semantic tag for the **bottom section** of the webpage.

Contains **contact details and copyright information**.

```
<footer>
    &copy; 2025 Department of Computer Science. All Rights Reserved.
</footer>
```

`&copy;` represents the **copyright symbol (©)**.

Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Department Page</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
            color: #333;
        }
        header {
            background: #007bff;
            color: white;
            text-align: center;
            padding: 20px;
            font-size: 24px;
        }
        nav {
            display: flex;
            justify-content: center;
            background: #0056b3;
            padding: 10px;
        }
        nav a {
            color: white;
            text-decoration: none;
            padding: 10px 20px;
            margin: 0 10px;
            font-size: 18px;
        }
        nav a:hover {
            background: #003d82;
            border-radius: 5px;
        }
        .container {
            width: 80%;
            margin: auto;
            padding: 20px;
            text-align: center;
        }
        .image-container {
            text-align: center;
            margin-top: 20px;
        }
        .image-container img {
            width: 300px;
            border-radius: 10px;
        }
        footer {
            background: #007bff;
            color: white;
            text-align: center;
            padding: 10px;
            position: fixed;
            bottom: 0;
            width: 100%;
        }
    </style>
</head>
<body>
    <header>
        Welcome to the Department of Computer Science
    </header>

    <nav>
        <a href="#about">About</a>
        <a href="#faculty">Faculty</a>
        <a href="#research">Research</a>
        <a href="#contact">Contact</a>
    </nav>

    <div class="container" id="about">
        <h2>About Our Department</h2>
        <p>The Department of Computer Science is dedicated to excellence in
teaching, research, and innovation. We offer undergraduate and postgraduate
programs that prepare students for industry and academia.</p>
    </div>

    <div class="container" id="faculty">
        <h2>Our Faculty</h2>
        <p>Our faculty consists of experienced professors and researchers
who guide students towards success in various fields of computing.</p>
    </div>

    <div class="container image-container">
        <h2>Department Image</h2>
        <img src="https://via.placeholder.com/300" alt="Department Image">
    </div>

    <div class="container" id="research">
        <h2>Research Areas</h2>
        <p>We focus on AI, machine learning, cybersecurity, and software
engineering, contributing to cutting-edge developments in computing.</p>
    </div>

    <div class="container" id="contact">
        <h2>Contact Us</h2>
        <p>Email: csdept@example.com | Phone: +1234567890</p>
    </div>

    <footer>
        &copy; 2025 Department of Computer Science. All Rights Reserved.
    </footer>
</body>
</html>
```

Conclusion:

This experiment demonstrated how to create a structured departmental webpage with background styling, images, navigation links, tables, and typography enhancements for better readability and presentation.

Experiment:-3

Title: Design and Implementation of a Simple Calculator Using JavaScript.

Aim: To develop a simple web-based calculator using JavaScript that can perform basic arithmetic operations: addition (sum), subtraction (difference), multiplication (product), and division (quotient).

Objectives:

To understand the fundamentals of JavaScript functions and event handling.

To implement a calculator that performs arithmetic operations.

To learn how to take user input, process it, and display results dynamically.

To apply DOM manipulation for real-time interaction.

To enhance programming skills in JavaScript, HTML, and CSS.

Materials Required:

A computer with a web browser

A text editor (e.g., VS Code, Notepad++, Sublime Text)

Basic knowledge of HTML, CSS, and JavaScript

Theory:

A calculator is a simple computational device that performs mathematical operations.In this experiment, we will create a basic web-based calculator using JavaScript that can add, subtract, multiply, and divide two numbers entered by the user.

Technologies Used:

HTML (HyperText Markup Language): Defines the structure of the calculator interface.

CSS (Cascading Style Sheets): Provides styling and improves the visual appeal of the calculator.

JavaScript: Adds functionality, processes user inputs, and displays results dynamically.

HTML Tags & JavaScript Concepts

1)HTML Tags:

- <input>: Accepts numbers from the user.
- <button>: Creates buttons for arithmetic operations.
- <div>: Displays the result.

2)JavaScript Concepts

2. DOM Manipulation: Used to capture user input and update the result dynamically.
3. Functions: Used to define arithmetic operations.
4. Event Listeners: Captures button clicks and triggers calculations.
5. parseFloat(): Converts string input to numbers for computation.

Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Calculator</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 50px;
        }
        .calculator {
            display: inline-block;
            padding: 20px;
            border: 2px solid #000;
            border-radius: 10px;
            background-color: #f4f4f4;
        }
        input, button {
            margin: 5px;
            padding: 10px;
            font-size: 16px;
        }
    </style>
</head>
<body>

    <h2>Simple Calculator</h2>
    <div class="calculator">
        <input type="number" id="num1" placeholder="Enter first number">
        <input type="number" id="num2" placeholder="Enter second number">
        <br>
        <button onclick="calculate('add')">+</button>
        <button onclick="calculate('subtract')">-</button>
        <button onclick="calculate('multiply')">×</button>
        <button onclick="calculate('divide')">÷</button>
        <br>
        <h3>Result: <span id="result">0</span></h3>
    </div>

    <script>
        function calculate(operation) {
            let num1 = parseFloat(document.getElementById("num1").value);
            let num2 = parseFloat(document.getElementById("num2").value);
            let result = 0;

            if (isNaN(num1) || isNaN(num2)) {
                document.getElementById("result").innerText = "Enter valid numbers";
                return;
            }

            switch (operation) {
                case "add":
                    result = num1 + num2;
                    break;
                case "subtract":
                    result = num1 - num2;
                    break;
                case "multiply":
                    result = num1 * num2;
                    break;
                case "divide":
                    if (num2 === 0) {
                        document.getElementById("result").innerText = "Cannot divide by zero";
                        return;
                    }
                    result = num1 / num2;
                    break;
            }

            document.getElementById("result").innerText = result;
        }
    </script>

</body>
</html>
```

Explanation of Code

2. **HTML Structure:**
    1. Two <input> fields for user input.
    2. Four <button> elements for arithmetic operations.
    3. A <span> element to display the result dynamically.
3. **CSS Styling:**
    1. The .calculator class creates a simple box-styled calculator.
    2. Buttons are styled for better visibility.
4. **JavaScript Functionality:**
    1. The calculate(operation) function fetches user input, performs the selected arithmetic operation, and updates the result.
    2. switch statement determines the operation type (addition, subtraction, multiplication, or division).
    3. parseFloat() converts input values from strings to numbers.
    4. document.getElementById("result").innerText updates the displayed result dynamically.

Conclusion:

A simple web-based calculator was successfully implemented using JavaScript. The program allows users to perform basic arithmetic operations through a user-friendly interface.

Experiment No-4

Title: Employee Form Validation Using JavaScript

Aim : To develop a JavaScript-based validation system for an HTML employee form that ensures the correctness of fields such as Name, Email, Mobile Number, Address, and Salary before submission.

Objectives:

To understand JavaScript's role in form validation.

To implement real-time validation for input fields.

To restrict invalid data entry in fields like email, phone number, and salary.

To enhance user experience by displaying error messages dynamically.

To learn how to use regular expressions (RegEx) for pattern matching in input validation.

Materials Required:

A computer with a web browser
A text editor (VS Code, Notepad++, Sublime Text)
Basic knowledge of HTML, CSS, and JavaScript

Theory:

1)Form Validation

Form validation is essential to prevent incorrect or malicious data entry before it is submitted to a server. It ensures:

Data Accuracy (e.g., email must follow a proper format)

Security (e.g., prevent SQL injection or script injection)

User Experience (e.g., guiding users with real-time error messages)

2)Types of Validation

Client-Side Validation (Using JavaScript)

Faster response, as checks happen before form submission.

Reduces unnecessary server load.

Example: Checking if an email contains '@' before submission.

Server-Side Validation (Using PHP, Node.js, etc.)

More secure, as it runs on the backend.

Ensures data integrity even if JavaScript is disabled.

Technologies Used:

1)HTML (HyperText Markup Language)

 Use:

- Defines the structure of the employee registration form.
- Contains input fields for Name, Email, Mobile Number, Address, and Salary.
- Uses `<form>`, `<input>`, `<button>`, and `<span>` elements.

 Why HTML?

6. Provides a structured and user-friendly interface.
7. Forms the backbone of the web page.

2) CSS (Cascading Style Sheets)

 Use:

5. Enhances the appearance of the form using styles such as padding, borders, colors, and error message styling.
6. Ensures responsive and readable text alignment.

 Why CSS?

2. Makes the form visually appealing and user-friendly.
3. Improves the readability of validation messages.

3) JavaScript (Client-Side Scripting Language)

 Use:

- Handles **real-time form validation** before submission.
- Uses **Regular Expressions (RegEx)** to validate Email and Mobile Number.
- Implements error handling using `document.getElementById()`.
- Uses **event handling (`onsubmit`)** to prevent invalid form submission.

 Why JavaScript?

- Provides instant feedback to users without needing server communication.
- Reduces server load by catching errors early.

- Enhances the overall user experience.

4) Regular Expressions (RegEx) for Validation

□ **Use:**

Validates **Email Format** (`/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}/` □)

Ensures **Mobile Number** starts with `6-9` and is exactly **10 digits long** (`/^[6-9]\d{9}$/` □)

□ **Why RegEx?**

Provides a powerful and efficient way to check patterns in user input.

Reduces manual string checking efforts.

HTML Tags & JavaScript Functions Used

1)HTML Elements:

<form>: Defines the employee registration form.

<input>: Used to enter values like name, email, phone number, etc.

<span>: Displays validation error messages.

<button>: Submits the form after validation.

2)JavaScript Functions & Concepts:

document.getElementById() → Fetches form elements.

RegExp (Regular Expressions) → Used to validate email and mobile numbers.

trim() → Removes extra spaces from input values.

isNaN() → Checks if a value is a valid number (for salary validation).

Event Handling (onsubmit) → Prevents form submission if validation fails.

Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Employee Form Validation</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 50px;
        }
        form {
            display: inline-block;
            text-align: left;
            padding: 20px;
            border: 2px solid #000;
            border-radius: 10px;
            background-color: #f4f4f4;
        }
        label, input {
            display: block;
            margin-bottom: 10px;
        }
        input {
            padding: 8px;
            width: 250px;
        }
        .error {
            color: red;
            font-size: 14px;
        }
        button {
            padding: 10px;
            font-size: 16px;
            cursor: pointer;
            background-color: green;
            color: white;
            border: none;
            border-radius: 5px;
        }
    </style>
</head>
<body>

    <h2>Employee Registration Form</h2>
    <form onsubmit="return validateForm()">
        <label>Name:</label>
        <input type="text" id="name">
        <span class="error" id="nameError"></span>

        <label>Email:</label>
        <input type="text" id="email">
        <span class="error" id="emailError"></span>
        <label>Mobile No.:</label>
        <input type="text" id="mobile">
        <span class="error" id="mobileError"></span>

        <label>Address:</label>
        <input type="text" id="address">
        <span class="error" id="addressError"></span>

        <label>Salary:</label>
        <input type="text" id="salary">
        <span class="error" id="salaryError"></span>

        <br><br>
        <button type="submit">Submit</button>
    </form>

    <script>
        function validateForm() {
            let isValid = true;

            // Get form values
            let name = document.getElementById("name").value.trim();
            let email = document.getElementById("email").value.trim();
            let mobile = document.getElementById("mobile").value.trim();
            let address = document.getElementById("address").value.trim();
            let salary = document.getElementById("salary").value.trim();

            // Regular expressions for validation
            let emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
            let mobilePattern = /^[6-9]\d{9}$/; // Indian 10-digit mobile numbers

            // Clear previous errors
            document.getElementById("nameError").innerText = "";
            document.getElementById("emailError").innerText = "";
            document.getElementById("mobileError").innerText = "";
            document.getElementById("addressError").innerText = "";
            document.getElementById("salaryError").innerText = "";

            // Name validation
            if (name === "") {
                document.getElementById("nameError").innerText = "Name is required";
                isValid = false;
            }

            // Email validation
            if (!emailPattern.test(email)) {
                document.getElementById("emailError").innerText = "Enter a valid email";
                isValid = false;
            }
```

```
            // Mobile number validation
            if (!mobilePattern.test(mobile)) {
                document.getElementById("mobileError").innerText = "Enter a valid 10-digit mobile number";
                isValid = false;
            }

            // Address validation
            if (address === "") {
                document.getElementById("addressError").innerText = "Address is required";
                isValid = false;
            }

            // Salary validation
            if (salary === "" || isNaN(salary) || salary <= 0) {
                document.getElementById("salaryError").innerText = "Enter a valid salary";
                isValid = false;
            }

            return isValid; // Prevents form submission if any field is invalid
        }
    </script>

</body>
</html>
```

Explanation of Code:

Form Elements: The form includes Name, Email, Mobile No, Address, and Salary fields.

Validation Logic:

Name: Ensures it is not empty.

Email: Uses a regular expression (RegEx) to check for a valid email format.

Mobile Number: Ensures it contains 10 digits and starts with 6-9.

Address: Checks if it is filled.

Salary: Ensures it is a positive number.

Error Messages: Displayed using `<span>` elements next to each field.

Return Value: The function returns `false` if any validation fails, preventing form submission.

Conclusion:

In this experiment, we successfully implemented client-side form validation using JavaScript. The validation ensures that users enter correct and meaningful data before submitting the form. This approach improves data accuracy, enhances security, and prevents server load from invalid submissions.

Experiment:-5

Title: JavaScript Functions to Find String Length and Reverse a Number

Aim: To develop an interactive HTML page that includes JavaScript functions to:

Calculate the length of a given string.

Reverse the digits of a given number.

Objectives:

Understand JavaScript functions and how to pass parameters.

Manipulate strings and numbers using built-in JavaScript methods.

Display the processed output dynamically on a webpage.

Materials Required:

4. A computer with a web browser
5. A text editor (VS Code, Notepad++, Sublime Text)
6. Basic knowledge of HTML, CSS, and JavaScript

Theory:

1)String Length Calculation:

7. The **length of a string** is determined using the `.length` property in JavaScript.

Example:

```
let str = "Hello";
console.log(str.length); // Output: 5
```

8. The **space** is also counted as a character.

2)Reverse a Number:

9. Steps to reverse a number:

Convert the number to a **string** (`toString()` method).

Use **`split('')`** to break it into an array of digits.

Reverse the array using **`reverse()`**.

Convert back to a **number** using `parseInt()`.

Example:

```
let num = 12345;
let reversedNum = parseInt(num.toString().split('').reverse().join(''));
console.log(reversedNum); // Output: 54321
```

Technology Used

1)HTML (HyperText Markup Language)

Provides the **structure** of the webpage.

Defines input fields for user input and buttons for triggering JavaScript functions.

Displays results dynamically using `<p>` elements.

2)CSS (Cascading Style Sheets)

Styles the webpage to make it visually appealing.

Enhances user experience using background colors, fonts, and button styles.

Helps position elements properly using margin, padding, and borders.

3)JavaScript (JS)

- Handles **user input** by fetching values from HTML elements.
- Implements **functions** for:
  - Finding the **length** of a string using `.length` property.
  - Reversing a **number's digits** using `split()`, `reverse()`, and `join()`.
- Updates the result dynamically on the webpage.

HTML Tags ,CSS & JavaScript Functions Used

HTML Form:

Two input fields (one for string, one for number).

Two buttons to trigger the JavaScript functions.

Two <p> tags for dynamically displaying results.

JavaScript Functions:

findStringLength()

Uses .length to find the length of the input string.

Displays the result in the <p> tag.

reverseNumber()

Converts the number into a string, reverses it, and converts it back.

Uses isNaN() to check for invalid inputs.

CSS Styling:

Forms have a **border, padding, and background color** for better UI.

Error handling prevents blank or invalid inputs.

Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>String Length & Number Reversal</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 50px;
        }
        .container {
            display: inline-block;
            text-align: left;
            padding: 20px;
            border: 2px solid #000;
            border-radius: 10px;
            background-color: #f4f4f4;
        }
        input {
            padding: 8px;
            width: 200px;
            margin-bottom: 10px;
        }
        button {
            padding: 10px;
            font-size: 16px;
            cursor: pointer;
            background-color: green;
            color: white;
            border: none;
            border-radius: 5px;
        }
        p {
            margin-top: 10px;
            font-size: 18px;
            color: blue;
        }
    </style>
</head>
<body>

    <h2>JavaScript String & Number Functions</h2>

    <div class="container">
        <h3>Find Length of String</h3>
        <input type="text" id="stringInput" placeholder="Enter a string">
        <button onclick="findStringLength()">Get Length</button>
        <p id="stringResult"></p>
    </div>

    <br><br>

    <div class="container">
        <h3>Reverse a Number</h3>
        <input type="number" id="numberInput" placeholder="Enter a number">
        <button onclick="reverseNumber()">Reverse</button>
        <p id="numberResult"></p>
    </div>

    <script>
        // Function to find string length
        function findStringLength() {
            let inputString = document.getElementById("stringInput").value;
            let length = inputString.length; // Get string length
            document.getElementById("stringResult").innerText = "Length of String: " + length;
        }

        // Function to reverse a number
        function reverseNumber() {
            let inputNum = document.getElementById("numberInput").value;
            if (isNaN(inputNum) || inputNum === "") {
                document.getElementById("numberResult").innerText = "Please enter a valid
number!";
                return;
            }

            let reversedNum = parseInt(inputNum.toString().split('').reverse().join(''));
            document.getElementById("numberResult").innerText = "Reversed Number: " +
reversedNum;
        }
    </script>

</body>
</html>
```

Explanation of Code:

Function to Find the Length of a String:

Fetches the user input from the text box (document.getElementById).
Uses the .length property to get the number of characters in the string.
Displays the output inside a <p> tag.

Function to Reverse a Number:-

Retrieves user input from the number field.
Validates the input using isNaN() to check if it's a valid number.
Converts the number to a string (toString()), splits it into an array of digits (split('')), reverses it (reverse()), and joins it back (join('')).
Converts the final string back to a number using parseInt().
Displays the reversed number in a <p> tag.

Conclusion

This experiment successfully implemented JavaScript functions to:

Find the length of a string using `.length` property.

Reverse the digits of a number using `.split()`, `.reverse()`, and `.join()`.

Experiment:-6

Title: Finding Prime Numbers Between Two Intervals Using JavaScript

Objective:

Understand the concept of **prime numbers** and their identification.
Use **JavaScript loops and conditions** to check prime numbers.
Take **user input** through an HTML form and process it using JavaScript.
Display the **output dynamically** on the webpage.

Aim:

To develop and demonstrate an HTML file that includes JavaScript to find and display all prime numbers between two given intervals.

Materials Required:

A computer with a web browser.

A text editor (e.g., VS Code, Notepad++, Sublime Text).

Basic knowledge of HTML and JavaScript and CSS.

Theory:

A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. To determine if a number is prime, it should not be divisible by any number other than 1 and itself. Efficient algorithms, such as checking divisibility up to the square root of the number, can be employed to enhance performance.

Examples of prime numbers: **2, 3, 5, 7, 11, 13, 17, etc.**

To find prime numbers between two numbers:

- Start from the **lower limit** and check every number up to the **upper limit**.
- For each number, check if it has **exactly two factors** (1 and itself).
- Use JavaScript's **loops and conditions** to perform these checks.

Technologies Used:

1)HTML (HyperText Markup Language):-

Used for structuring the webpage, including input fields, buttons, and result display areas.

2)CSS (Cascading Style Sheets):-

Used to style the webpage, enhance user experience, and make the UI visually appealing

3)JavaScript (JS):-

The core logic for finding prime numbers, handling user input, validating numbers, and dynamically updating results.

4)Math Functions in JavaScript:-

`Math.sqrt()` is used to optimize prime number checking.

Program:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prime Numbers Finder</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 50px;
        }
        .container {
            display: inline-block;
            text-align: left;
            padding: 20px;
            border: 2px solid #000;
            border-radius: 10px;
            background-color: #f4f4f4;
        }
        input {
            padding: 8px;
            width: 200px;
            margin-bottom: 10px;
        }
        button {
            padding: 10px;
            font-size: 16px;
            cursor: pointer;
            background-color: green;
            color: white;
            border: none;
            border-radius: 5px;
        }
        p {
            margin-top: 10px;
            font-size: 18px;
            color: blue;
        }
    </style>
</head>
<body>

    <h2>Find Prime Numbers Between Two Numbers</h2>

    <div class="container">
        <h3>Enter Range</h3>
        <input type="number" id="startNum" placeholder="Enter Start Number">
        <input type="number" id="endNum" placeholder="Enter End Number">
        <button onclick="findPrimes()">Find Primes</button>
        <p id="result"></p>
    </div>

    <script>
        function findPrimes() {
            let start = parseInt(document.getElementById("startNum").value);
            let end = parseInt(document.getElementById("endNum").value);
            let primes = [];

            if (isNaN(start) || isNaN(end) || start < 2 || end < 2 || start > end) {
                document.getElementById("result").innerText = "Please enter valid numbers (start
≥ 2, end ≥ start).";
                return;
            }

            for (let num = start; num <= end; num++) {
                let isPrime = true;

                for (let i = 2; i <= Math.sqrt(num); i++) {
                    if (num % i === 0) {
                        isPrime = false;
                        break;
                    }
                }

                if (isPrime && num > 1) {
```

```
                primes.push(num);
            }
        }

        document.getElementById("result").innerText =
            primes.length > 0 ? "Prime Numbers: " + primes.join(", ") : "No prime numbers
found.";
    }
  </script>

</body>
</html>
```

Explanation of Code:

1.Getting User Input

document.getElementById("startNum").value fetches the **starting number** entered by the user.

document.getElementById("endNum").value fetches the **ending number** entered by the user.

2.Checking Prime Numbers

A **loop runs from the starting number to the ending number**.

A **nested loop** checks whether each number has **exactly two factors (1 and itself)**.

If a number is **prime**, it is added to the result list.

3. **Displaying the Output**

The list of prime numbers is displayed inside a <p> tag dynamically.

Conclusion:

The experiment successfully demonstrates how to use JavaScript within an HTML file to find and display all prime numbers between a user-specified range. The implementation includes input validation and efficiently determines prime numbers using optimized algorithms.

PHP Programming Lab Manual

Lab Objectives

2. To understand the basic syntax of PHP.
3. To learn how to define variables and constants in PHP.
4. To explore PHP data types, operators, and expressions.
5. To handle HTML forms using PHP, including capturing form data and dealing with multi-value fields.
6. To implement redirection after form submission.
7. To understand and use PHP sessions for maintaining user state.

What is PHP?

PHP is an acronym for "PHP: Hypertext Preprocessor"

PHP is a widely-used, open source scripting language

PHP scripts are executed on the server

PHP is free to download and use

What is a PHP File?

PHP files can contain text, HTML, CSS, JavaScript, and PHP code

PHP code is executed on the server, and the result is returned to the browser as plain HTML

PHP files have extension "`.php`"

What Can PHP Do?

PHP can generate dynamic page content

PHP can create, open, read, write, delete, and close files on the server

PHP can collect form data

PHP can send and receive cookies

PHP can add, delete, modify data in your database

PHP can be used to control user-access

PHP can encrypt data

With PHP you are not limited to output HTML. You can output images or PDF files. You can also output any text, such as XHTML and XML.

Why PHP?

PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)

PHP is compatible with almost all servers used today (Apache, IIS, etc.)

PHP supports a wide range of databases

PHP is free. Download it from the official PHP resource: www.php.net

PHP is easy to learn and runs efficiently on the server side

PHP, a powerful server-side scripting language used in web development. It's simplicity and ease of use makes it an ideal choice for beginners and experienced developers. This article provides an overview of PHP syntax. PHP scripts can be written anywhere in the document within PHP tags along with normal HTML.

Basic PHP Syntax
PHP code is executed between PHP tags, allowing the integration of PHP code within HTML. The most common PHP tag is **<?php … ?>**, which is used to enclose PHP code. The **<?php ….?>** is called **Escaping to PHP**.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

```
<!DOCTYPE html>

<html>

<body>

<h1>My first PHP page</h1>

<?php

echo "Hello World!";

?>

</body>
```

```
</html>
```

Note: PHP statements end with a semicolon (`;`).

PHP Case Sensitivity

In PHP, keywords (e.g. `if`, `else`, `while`, `echo`, etc.), classes, functions, and user-defined functions are not case-sensitive.In the example below, all three echo statements below are equal and legal:

Example

`ECHO` is the same as `echo`:

```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>

</body>
</html>
```

Output:-

```
Hello World!
Hello World!
Hello World!
```

PHP Variables

PHP Variables are one of the most fundamental concepts in programming. It is used to store data that can be accessed and manipulated within your code. Variables in PHP are easy to use, dynamically typed (meaning that you do not need to declare their type explicitly), and essential for creating dynamic, interactive web applications.

Declaring Variables in PHP

A variable can have a short name (like $x and $y) or a more descriptive name (`$age`, `$carname`, `$total_volume`).

Rules for PHP variables:

A variable starts with the $ sign, followed by the name of the variable

A variable name must start with a letter or the underscore character

A variable name cannot start with a number

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Variable names are case-sensitive (`$age` and `$AGE` are two different variables)

Remember that PHP variable names are case-sensitive!

Output Variables

The PHP `echo` statement is often used to output data to the screen.
The following example will show how to output text and a variable:

```
$txt = "W3Schools.com";

echo "I love $txt!";
```

Output:

```
I love W3Schools.com!
```

PHP Constants

Constants are like variables, except that once they are defined they can not be changed or undefined. constant is an identifier (name) for a simple value. The value cannot be changed during the script. A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

Create a PHP Constant

To create a constant, use the `define()` function.

Syntax:- `define(name, value);`

Parameters:

name: Specifies the name of the constant

value: Specifies the value of the constant

Example

Create a constant with a case-sensitive **name:**

```
define("GREETING", "Welcome to W3Schools.com!");

echo GREETING;
```

Output:-

```
Welcome to W3Schools.com!
```

PHP const Keyword

You can also create a constant by using the `const` keyword.

Example

Create a **case-sensitive** constant with the `const` keyword:

```
const MYCAR = "Volvo";

echo MYCAR;
```

OUTPUT:- Volvo

const vs. define()

`const` cannot be created inside another block scope, like inside a function or inside an `if` statement.

`define` can be created inside another block scope.

PHP Constant Arrays

From PHP7, you can create an Array constant using the `define()` function.

Example

Create an Array constant:

```
efine("cars", [

  "Alfa Romeo",

  "BMW",

  "Toyota"

]);

echo cars[0];
```

Output:- Alfa Romeo

Constants are Global

Constants are automatically global and can be used across the entire script.

Example

This example uses a constant inside a function, even if it is defined outside the function:

```
define("GREETING", "Welcome to W3Schools.com!");



function myTest() {

  echo GREETING;

}



myTest();
```

output:- Welcome to W3Schools.com!

PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

String

Integer

Float (floating point numbers - also called double)

Boolean

Array

Object

NULL

Resource

Getting the Data Type

You can get the data type of any object by using the `var_dump()` function.

ExampleGet your own PHP Server

The `var_dump()` function returns the data type and the value:

```
x = 5;

var_dump($x);
```

Output:- int(7)

PHP String

A string is a sequence of characters, like "Hello world!".A string can be any text inside quotes. You can use single or double quotes:

```
$x = "Hello world!";
```

```
$y = 'Hello world!';


var_dump($x);

echo "<br>";

var_dump($y);
```

Output:-

```
string(12) "Hello world!"
string(12) "Hello world!"
```

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

An integer must have at least one digit

An integer must not have a decimal point

An integer can be either positive or negative

Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example `$x` is an integer. The PHP `var_dump()` function returns the data type and value:

Example

```
$x = 5985;

var_dump($x);
```

Output:- int(5985)

PHP Float-

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example `$x` is a float. The PHP `var_dump()` function returns the data type and value:

Example

```
$x = 10.365;

var_dump($x);
```

Output:- float(10.365)

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

Example

```
$x = true;

var_dump($x);
```

Output:- bool(true)

Booleans are often used in conditional testing.

PHP Array

An array stores multiple values in one single variable.

In the following example `$cars` is an array. The PHP `var_dump()` function returns the data type and value:

Example

```
$cars = array("Volvo","BMW","Toyota");

var_dump($cars);
```

Output:

```
array(3) {
  [0]=>
  string(5) "Volvo"
```

```
 [1]=>
 string(3) "BMW"
 [2]=>
 string(6) "Toyota"
}
```

PHP Object

Classes and objects are the two main aspects of object-oriented programming.A class is a template for objects, and an object is an instance of a class. When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Let's assume we have a class named `Car` that can have properties like model, color, etc. We can define variables like `$model`, `$color`, and so on, to hold the values of these properties.

When the individual objects (Volvo, BMW, Toyota, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

If you create a `__construct()` function, PHP will automatically call this function when you create an object from a class.

Example

```
class Car {

  public $color;

  public $model;

  public function __construct($color, $model) {

    $this->color = $color;

    $this->model = $model;

  }

  public function message() {

    return "My car is a " . $this->color . " " . $this->model . "!";

  }

}
```

```
$myCar = new Car("red", "Volvo");

var_dump($myCar);
```

Output:_ object(Car)#1 (2) { ["color"]=> string(3) "red" ["model"]=> string(5) "Volvo" }

PHP NULL Value

Null is a special data type which can have only one value: NULL. A variable of data type NULL is a variable that has no value assigned to it.

Tip: If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

Example

```
$x = "Hello world!";

$x = null;

var_dump($x);
```

Output:- NULL

Change Data Type

If you assign an integer value to a variable, the type will automatically be an integer.If you assign a string to the same variable, the type will change to a string:

Example

```
$x = 5;

var_dump($x);



$x = "Hello";

var_dump($x);
```

Output :-

```
int(5)
string(5) "Hello"
```

Line breaks were added for better readability.

If you want to change the data type of an existing variable, but not by changing the value, you can use casting.

Casting allows you to change data type on variables:

Example

```
$x = 5;

$x = (string) $x;

var_dump($x);
```

Output :- string(1) "5"

PHP Operators

Operators are used to performing operations on some values. In other words, we can describe operators as something that takes some values, performs some operation on them, and gives a result. From example, "1 + 2 = 3" in this expression '+' is an operator. It takes two values 1 and 2, performs an addition operation on them to give 3.

Just like any other programming language, PHP also supports various types of operations like arithmetic operations(addition, subtraction, etc), logical operations(AND, OR etc), Increment/Decrement Operations, etc. Thus, PHP provides us with many operators to perform such operations on various operands or variables, or values. These operators are nothing but symbols needed to perform operations of various types. Given below are the various groups of operators:

10. Arithmetic Operators
8. Logical or Relational Operators
7. Comparison Operators
- Conditional or Ternary Operators
- Assignment Operators

Spaceship Operators (Introduced in PHP 7)
Array Operators
Increment/Decrement Operators
String Operators

Let us now learn about each of these operators in detail.

Arithmetic Operators:

The arithmetic operators are used to perform simple mathematical operations like addition, subtraction, multiplication, etc. Below is the list of arithmetic operators along with their syntax and operations in PHP.

Operator	***Name***	***Syntax***	***Operation***
+	Addition	$x + $y	Sum the operands
–	Subtraction	$x – $y	Differences the operands
*	Multiplication	$x * $y	Product of the operands
/	Division	$x / $y	The quotient of the operands
**	Exponentiation	$x ** $y	$x raised to the power $y
%	Modulus	$x % $y	The remainder of the operands

Note: The exponentiation has been introduced in PHP 5.6.

Example: This example explains the arithmetic operator in PHP.

```php
<?php
// Define two numbers
$x = 10;
$y = 3;

// Addition
echo "Addition: " . ($x + $y) . "\n";

// Subtraction
echo "Subtraction: " . ($x - $y) . "\n";

// Multiplication
echo "Multiplication: " . ($x * $y) . "\n";

// Division
echo "Division: " . ($x / $y) . "\n";

// Exponentiation
echo "Exponentiation: " . ($x ** $y) . "\n";

// Modulus
echo "Modulus: " . ($x % $y) . "\n";
?>
```

Output

```
Addition: 13

Subtraction: 7

Multiplication: 30

Division: 3.3333333333333

Exponentiation: 1000

Modulus: 1
```

Logical or Relational Operators:

These are basically used to operate with conditional statements and expressions. Conditional statements are based on conditions. Also, a condition can either be met or cannot be met so the result of a conditional statement can either be true or false. Here are the logical operators along with their syntax and operations in PHP.

Operator	*Name*	*Syntax*	*Operation*
and	Logical AND	$x and $y	True if both the operands are true else false
or	Logical OR	$x or $y	True if either of the operands is true else false
xor	Logical XOR	$x xor $y	True if either of the operands is true and false if both are true
&&	Logical AND	$x && $y	True if both the operands are true else false
||	Logical OR	$x || $y	True if either of the operands is true else false

Operator	***Name***	***Syntax***	***Operation***
!	Logical NOT	!$x	True if $x is false

Example: This example describes the logical & relational operator in PHP.

```
<?php
$x = 50;
$y = 30;
  if ($x == 50 and $y == 30)
      echo "and Success \n";

  if ($x == 50 or $y == 20)
      echo "or Success \n";

  if ($x == 50 xor $y == 20)
      echo "xor Success \n";

  if ($x == 50 && $y == 30)
      echo "&& Success \n";

  if ($x == 50 || $y == 20)
      echo "|| Success \n";

  if (!$z)
      echo "! Success \n";
?>
```

Output:

```
and Success
or Success
xor Success
&& Success
|| Success
! Success
```

Comparison Operators:

These operators are used to compare two elements and outputs the result in boolean form. Here are the comparison operators along with their syntax and operations in PHP

.

Operator	**Name**	**Syntax**	**Operation**

Operator	Name	Syntax	Operation
==	Equal To	$x == $y	Returns True if both the operands are equal
!=	Not Equal To	$x != $y	Returns True if both the operands are not equal
<>	Not Equal To	$x <> $y	Returns True if both the operands are unequal
===	Identical	$x === $y	Returns True if both the operands are equal and are of the same type
!==	Not Identical	$x == $y	Returns True if both the operands are unequal and are of different types
<	Less Than	$x < $y	Returns True if $x is less than $y
>	Greater Than	$x > $y	Returns True if $x is greater than $y
<=	Less Than or Equal To	$x <= $y	Returns True if $x is less than or equal to $y
>=	Greater Than or Equal To	$x >= $y	Returns True if $x is greater than or equal to $y

Example: This example describes the comparison operator in PHP.

```
<?php
  $a = 80;
  $b = 50;
  $c = "80";

  // Here var_dump function has been used to
  // display structured information. We will learn
  // about this function in complete details in further
  // articles.
  var_dump($a == $c) + "\n";
  var_dump($a != $b) + "\n";
  var_dump($a <> $b) + "\n";
  var_dump($a === $c) + "\n";
  var_dump($a !== $c) + "\n";
  var_dump($a < $b) + "\n";
  var_dump($a > $b) + "\n";
  var_dump($a <= $b) + "\n";
  var_dump($a >= $b);
?>
```

Output:

```
bool(true)
bool(true)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
```

Conditional or Ternary Operators:

These operators are used to compare two values and take either of the results simultaneously, depending on whether the outcome is TRUE or FALSE. These are also used as a shorthand notation for *if…else* statement that we will read in the article on decision making.

Syntax:

```
$var = (condition)? value1 : value2;
```

Here, the condition will either evaluate as true or false. If the condition evaluates to True, then value1 will be assigned to the variable $var otherwise value2 will be assigned to it.

Operator	***Name***	***Operation***
?:	Ternary	If the condition is true? then $x : or else $y. This means that if the condition is true then the left result of the colon is accepted otherwise the result is on right.

Example: This example describes the Conditional or Ternary operators in PHP.

```
<?php
  $x = -12;
  echo ($x > 0) ? 'The number is positive' : 'The number is negative';
?>
```

Output:

```
The number is negative
```

Assignment Operators:

These operators are used to assign values to different variables, with or without mid-operations. Here are the assignment operators along with their syntax and operations, that PHP provides for the operations.

Operator	***Name***	***Syntax***	***Operation***
=	Assign	$x = $y	Operand on the left obtains the value of the operand on the right
+=	Add then Assign	$x += $y	Simple Addition same as $x = $x + $y
-=	Subtract then Assign	$x -= $y	Simple subtraction same as $x = $x – $y
*=	Multiply then Assign	$x *= $y	Simple product same as $x = $x * $y
/=	Divide then Assign (quotient)	$x /= $y	Simple division same as $x = $x / $y
%=	Divide then Assign (remainder)	$x %= $y	Simple division same as $x = $x % $y

Example: This example describes the assignment operator in PHP.

```
<?php
  // Simple assign operator
  $y = 75;
  echo $y, "\n";

  // Add then assign operator
  $y = 100;
  $y += 200;
  echo $y, "\n";
  // Subtract then assign operator
  $y = 70;
  $y -= 10;
  echo $y, "\n";

  // Multiply then assign operator
  $y = 30;
  $y *= 20;
  echo $y, "\n";

  // Divide then assign(quotient) operator
  $y = 100;
  $y /= 5;
  echo $y, "\n";

  // Divide then assign(remainder) operator
  $y = 50;
  $y %= 5;
  echo $y;
?>
```

Output:

```
75
300
60
600
20
0
```

Array Operators:

These operators are used in the case of arrays. Here are the array operators along with their syntax and operations, that PHP provides for the array operation.

Operator	***Name***	***Syntax***	***Operation***
+	Union	$x + $y	Union of both i.e., $x and $y
==	Equality	$x == $y	Returns true if both has same key-value pair
!=	Inequality	$x != $y	Returns True if both are unequal
===	Identity	$x === $y	Returns True if both have the same key-value pair in the same order and of the same type

Operator	***Name***	***Syntax***	***Operation***
!==	Non-Identity	$x !== $y	Returns True if both are not identical to each other
<>	Inequality	$x <> $y	Returns True if both are unequal

Example: This example describes the array operation in PHP.

```
<?php
  $x = array("k" => "Car", "l" => "Bike");
  $y = array("a" => "Train", "b" => "Plane");

  var_dump($x + $y);
  var_dump($x == $y) + "\n";
  var_dump($x != $y) + "\n";
  var_dump($x <> $y) + "\n";
  var_dump($x === $y) + "\n";
  var_dump($x !== $y) + "\n";
?>
```

Output:

```
array(4) {
  ["k"]=>
  string(3) "Car"
  ["l"]=>
  string(4) "Bike"
  ["a"]=>
  string(5) "Train"
  ["b"]=>
  string(5) "Plane"
}
bool(false)
bool(true)
bool(true)
bool(false)
bool(true)
```

Increment/Decrement Operators:

These are called the unary operators as they work on single operands. These are used to increment or decrement values.

Operator	***Name***	***Syntax***	***Operation***
++	Pre-Increment	++$x	First increments $x by one, then return $x
—	Pre-Decrement	–$x	First decrements $x by one, then return $x
++	Post-Increment	$x++	First returns $x, then increment it by one
—	Post-Decrement	$x–	First returns $x, then decrement it by one

Example: This example describes the Increment/Decrement operators in PHP.

```
<?php
  $x = 2;
  echo ++$x, " First increments then prints \n";
  echo $x, "\n";

  $x = 2;
  echo $x++, " First prints then increments \n";
  echo $x, "\n";

  $x = 2;
  echo --$x, " First decrements then prints \n";
  echo $x, "\n";

  $x = 2;
  echo $x--, " First prints then decrements \n";
  echo $x;
?>
```

Output:

```
3 First increments then prints
3
2 First prints then increments
3
1 First decrements then prints
1
2 First prints then decrements
1
```

String Operators:

This operator is used for the concatenation of 2 or more strings using the concatenation operator (‘.’). We can also use the concatenating assignment operator (‘.=’) to append the argument on the right side to the argument on the left side.

Operator	***Name***	***Syntax***	***Operation***
.	Concatenation	$x.$y	Concatenated $x and $y
.=	Concatenation and assignment	$x.=$y	First concatenates then assigns, same as $x = $x.$y

Example: This example describes the string operator in PHP.

```
<?php
  $x = "Geeks";
  $y = "for";
  $z = "Geeks!!!";
  echo $x . $y . $z, "\n";
  $x .= $y . $z;
  echo $x;
?>
```

Output:

```
GeeksforGeeks!!!
GeeksforGeeks!!!
```

Spaceship Operators:

PHP 7 has introduced a new kind of operator called spaceship operator. The spaceship operator or combined comparison operator is denoted by “<=>“. These operators are used to compare values but instead of returning the boolean results, it returns integer values. If both the operands are equal, it returns 0. If the right operand is greater, it returns -1. If the left operand is greater, it returns 1. The following table shows how it works in detail:

Operator	***Syntax***	***Operation***
$x < $y	$x <=> $y	Identical to -1 (right is greater)
$x > $y	$x <=> $y	Identical to 1 (left is greater)

Operator	***Syntax***	***Operation***
$x <= $y	$x <=> $y	Identical to -1 (right is greater) or identical to 0 (if both are equal)
$x >= $y	$x <=> $y	Identical to 1 (if left is greater) or identical to 0 (if both are equal)
$x == $y	$x <=> $y	Identical to 0 (both are equal)
$x != $y	$x <=> $y	Not Identical to 0

Example: This example illustrates the use of the spaceship operator in PHP.

```
<?php
  $x = 50;
  $y = 50;
  $z = 25;

  echo $x <=> $y;
  echo "\n";

  echo $x <=> $z;
  echo "\n";

  echo $z <=> $y;
  echo "\n";

  // We can do the same for Strings
  $x = "Ram";
  $y = "Krishna";

  echo $x <=> $y;
  echo "\n";

  echo $x <=> $y;
  echo "\n";

  echo $y <=> $x;
?>
```

Output:

```
0
1
-1
```

```
1
1
-1
```

PHP Form Handling

The PHP superglobals $_GET and $_POST are used to collect form-data.

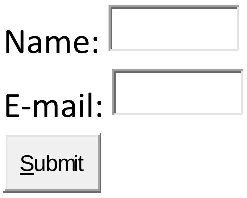
PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example

```
<html>

<body>



<form action="welcome.php" method="POST">

Name: <input type="text" name="name"><br>

E-mail: <input type="text" name="email"><br>

<input type="submit">

</form>



</body>

</html>
```

Output:-



Name:

E-mail:

hen the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables.

The "welcome.php" looks like this:

```
<html>

<body>



Welcome <?php echo $_POST["name"]; ?><br>

Your email address is: <?php echo $_POST["email"]; ?>



</body>

</html>
```

The output could be something like this:

```
Welcome John

Your email address is john.doe@example.com
```

The same result could also be achieved using the HTTP GET method:

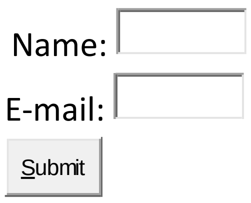
Example

Same example, but the method is set to GET instead of POST:

```
<html>

<body>



<form action="welcome_get.php" method="GET">

Name: <input type="text" name="name"><br>

E-mail: <input type="text" name="email"><br>

<input type="submit">

</form>
```

```
</body>

</html>
```

Output:



and "welcome_get.php" looks like this:

```
<html>

<body>



Welcome <?php echo $_GET["name"]; ?><br>

Your email address is: <?php echo $_GET["email"]; ?>



</body>

</html>
```

The code above is quite simple, and it does not include any validation.

You need to validate form data to protect your script from malicious code.

GET vs. POST

Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the URL parameters.

$_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

PHP Form Validation

Think SECURITY when processing PHP forms!

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:

The validation rules for the form above are as follows:.

Field	**Validation Rules**
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

First we will look at the plain HTML code for the form:

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea.

The HTML code looks like this:

```
Name: <input type="text" name="name">

E-mail: <input type="text" name="email">

Website: <input type="text" name="website">

Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

```
Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other
```

The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

When the form is submitted, the form data is sent with method="post".

What is the `$_SERVER["PHP_SELF"]` variable?

The `$_SERVER["PHP_SELF"]` is a super global variable that returns the filename of the currently executing script.

So, the `$_SERVER["PHP_SELF"]` sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the `htmlspecialchars()` function?

The `htmlspecialchars()` function converts special characters into HTML entities. This means that it will replace HTML characters
like < and > with `&lt;` and `&gt;`. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Warning!

The `$_SERVER["PHP_SELF"]` variable can be used by hackers!

If PHP_SELF is used in your page then a user can enter a slash / and then some Cross Site Scripting (XSS) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

So far, so good.

However, consider that a user enters the following URL in the address bar:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%
3C/script%3E
```

In this case, the above code will be translated to:

```
<form method="post"
action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

`$_SERVER["PHP_SELF"]` exploits can be avoided by using the `htmlspecialchars()` function.

The form code should look like this:

```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The `htmlspecialchars()` function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post"
action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/scrip
t&gt;">
```

The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's `htmlspecialchars()` function.

When we use the `htmlspecialchars()` function; then if a user tries to submit the following in a text field:

`<script>location.href('http://www.hacked.com')</script>`

- this would not be executed, because it would be saved as HTML escaped code, like this:

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

- Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP `trim()` function)

- Remove backslashes \ from the user input data (with the PHP `stripslashes()` function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function `test_input()`.

Now, we can check each `$_POST` variable with the `test_input()` function, and the script looks like this:

ExampleGet your own PHP Server

```
// define variables and set to empty values

$name = $email = $gender = $comment = $website = "";



if ($_SERVER["REQUEST_METHOD"] == "POST") {

  $name = test_input($_POST["name"]);

  $email = test_input($_POST["email"]);

  $website = test_input($_POST["website"]);

  $comment = test_input($_POST["comment"]);

  $gender = test_input($_POST["gender"]);

}



function test_input($data) {

  $data = trim($data);

  $data = stripslashes($data);

  $data = htmlspecialchars($data);

  return $data;

}
```

Run Example »

Notice that at the start of the script, we check whether the form has been submitted using `$_SERVER["REQUEST_METHOD"]`. If the `REQUEST_METHOD` is `POST`, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

The next step is to make input fields required and create error messages if needed.

PHP Forms - Required Fields

his chapter shows how to make input fields required and create error messages if needed.

PHP - Required Fields

From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

Field	**Validation Rules**
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL

Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

In the previous chapter, all input fields were optional.

In the following code we have added some new variables: `$nameErr`, `$emailErr`, `$genderErr`, and `$websiteErr`. These error variables will hold error messages for the required fields. We have also added an `if else` statement for each `$_POST` variable. This checks if the `$_POST` variable is empty (with the PHP `empty()` function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the `test_input()` function:

```
// define variables and set to empty values

$nameErr = $emailErr = $genderErr = $websiteErr = "";

$name = $email = $gender = $comment = $website = "";



if ($_SERVER["REQUEST_METHOD"] == "POST") {

  if (empty($_POST["name"])) {

    $nameErr = "Name is required";

  } else {

    $name = test_input($_POST["name"]);

  }



  if (empty($_POST["email"])) {

    $emailErr = "Email is required";

  } else {

    $email = test_input($_POST["email"]);

  }
  if (empty($_POST["website"])) {

    $website = "";

  } else {

    $website = test_input($_POST["website"]);

  }


  if (empty($_POST["comment"])) {

    $comment = "";

  } else {

    $comment = test_input($_POST["comment"]);

  }


  if (empty($_POST["gender"])) {

    $genderErr = "Gender is required";

  } else {

    $gender = test_input($_POST["gender"]);

  }

}
```

PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

Example[Get your own PHP Server]

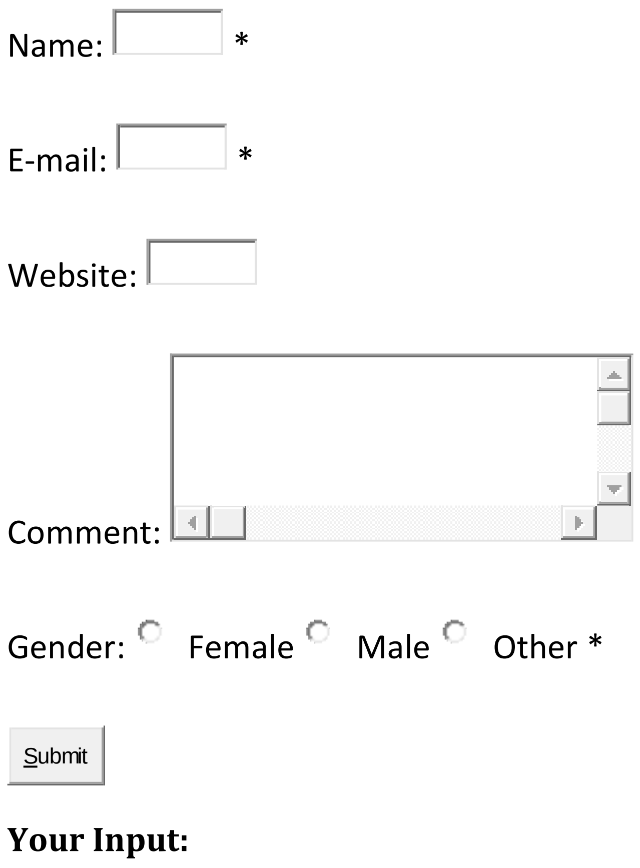
```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

```
Name: <input type="text" name="name">

<span class="error">* <?php echo $nameErr;?></span>

<br><br>

E-mail:

<input type="text" name="email">

<span class="error">* <?php echo $emailErr;?></span>

<br><br>

Website:

<input type="text" name="website">

<span class="error"><?php echo $websiteErr;?></span>

<br><br>

Comment: <textarea name="comment" rows="5" cols="40"></textarea>

<br><br>

Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other

<span class="error">* <?php echo $genderErr;?></span>

<br><br>

<input type="submit" name="submit" value="Submit">


</form>
```

Output:-

PHP Form Validation Example

* required field


Name: *

E-mail: *

Website:

Comment:

Gender: Female Male Other *

Submit

Your Input:

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".

PHP Forms - Validate E-mail and URL

This chapter shows how to validate names, e-mails, and URLs.

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters, dashes, apostrophes and whitespaces. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);

if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's `filter_var()` function.

In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);

if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {

  $emailErr = "Invalid email format";

}
```

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);

if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {

  $websiteErr = "Invalid URL";

}
```

PHP - Validate Name, E-mail, and URL

Now, the script looks like this:

ExampleGet your own PHP Server

```
// define variables and set to empty values
```

```
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);

    // check if URL address syntax is valid (this regular expression
also allows dashes in the URL)

    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {

      $websiteErr = "Invalid URL";

    }

  }


  if (empty($_POST["comment"])) {

    $comment = "";

  } else {

    $comment = test_input($_POST["comment"]);

  }


  if (empty($_POST["gender"])) {

    $genderErr = "Gender is required";

  } else {

    $gender = test_input($_POST["gender"]);

  }

}
```

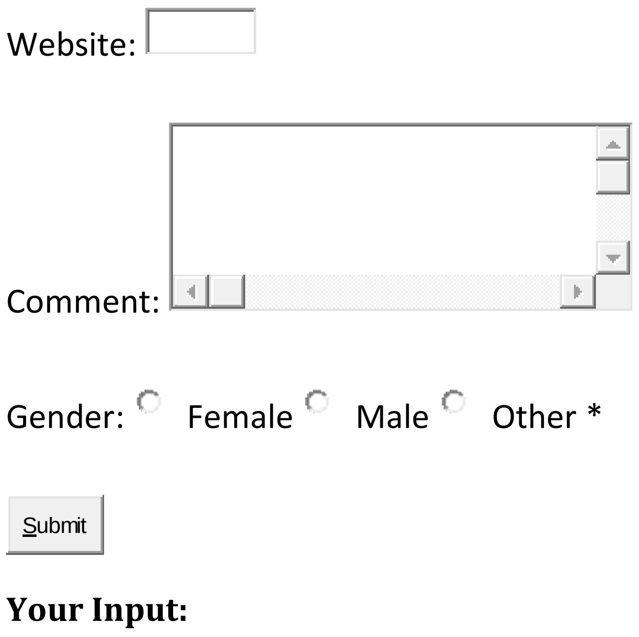
Output:-

PHP Form Validation Example

* required field

Name: *

E-mail: *



Website:

Comment:

Gender: Female Male Other *

Submit

Your Input:

The next step is to show how to prevent the form from emptying all the input fields when the user submits the form.

PHP Complete Form Example

PHP - Keep The Values in The Form

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the `<textarea>` and `</textarea>` tags. The little script outputs the value of the `$name`, `$email`, `$website`, and `$comment` variables.

Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

```
Name: <input type="text" name="name" value="<?php echo $name;?>">



E-mail: <input type="text" name="email" value="<?php echo $email;?>">



Website: <input type="text" name="website" value="<?php echo
$website;?>">
```

```
Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>


Gender:

<input type="radio" name="gender"

<?php if (isset($gender) && $gender=="female") echo "checked";?>

value="female">Female

<input type="radio" name="gender"

<?php if (isset($gender) && $gender=="male") echo "checked";?>

value="male">Male

<input type="radio" name="gender"

<?php if (isset($gender) && $gender=="other") echo "checked";?>

value="other">Other
```

PHP - Complete Form Example

Here is the complete code for the PHP Form Validation Example:

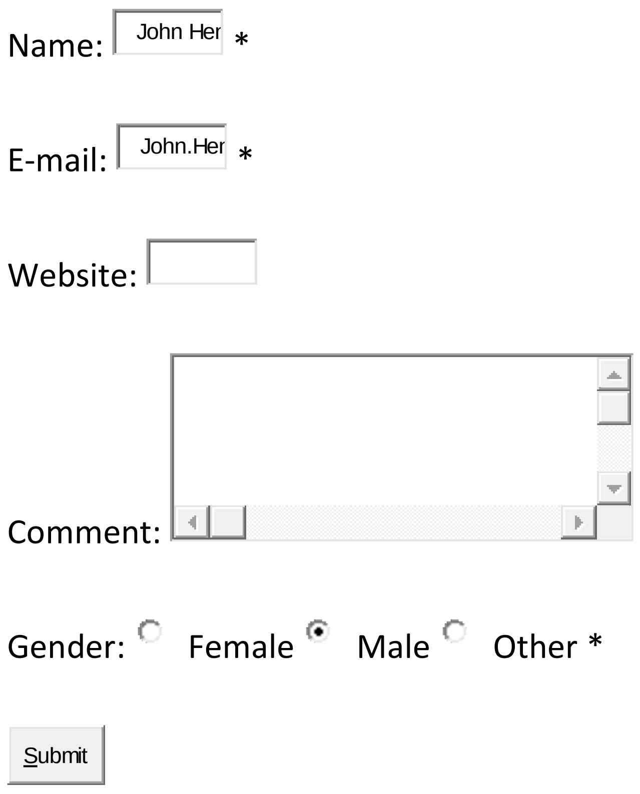
```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression
also allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

```
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_S
ELF"]);?>">
  Name: <input type="text" name="name" value="<?php echo $name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?php echo $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?php echo $website
;?>">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comm
ent;?></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="female") echo "checked";?> value="female">Female
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="male") echo "checked";?> value="male">Male
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="other") echo "checked";?> value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

Output:

PHP Form Validation Example

* required field



Name: John Her *

E-mail: John.Her *

Website:

Comment:

Gender: Female [✓ Male] Other *

Submit

Your Input:

John Henry
John.Henry@kbpcoes.edu.in

male

PHP - Dealing with Multi-Value Fields

Introduction

The following form fields are capable of sending multiple values to the server:

```
<label for="favoriteWidgets" >What are your favorite widgets?</label
>
<select name="favoriteWidgets" id="favoriteWidgets" size="3" multipl
e="multiple" >
    <option value="superWidget" >The SuperWidget</option>
    <option value="megaWidget" >The MegaWidget</option>
    <option value="wonderWidget" >The WonderWidget</option>
</select>


<label for="newsletterWidgetTimes" >Do you want to receive our 'Phon
eTimes'newsletter?</label>
<input type="checkbox" name="newsletter"  id="newsletterWidgetTimes
" value="widgetTimes" />
<label for="newsletterFunWithWidgets" >Do you want to receive our 'F
un with Widgets'newsletter?</label>
<input type="checkbox" name="newsletter" id="newsletterFunWithWidget
s" value="funWithWidgets" />
```

The first f

values (widgetTimes and funWithWidgets). If the user checks both checkboxes then both values, widgetTimes and funWithWidgets, are sent to the server under the newsletter field name.

To handle multi-value fields in your PHP scripts, add square brackets ([]) after the field name in your HTML form.

Then, when the PHP engine sees a submitted form field name with square brackets at the end, it creates a nested array of values within the $_GET or $_POST (and $_REQUEST) superglobal array, rather than a single value.

You can then pull the individual values out of that nested array.

So you might create a multi-select list control as follows:

```
<select name="favoriteWidgets[]"  id="favoriteWidgets"  size
="3"  multiple="multiple" ...  < /select >
```

You'd then retrieve the array containing the submitted field values as follows:

```
$favoriteWidgetValuesArray = $_GET[" favoriteWidgets" ];  //
 If using get method
$favoriteWidgetValuesArray = $_POST[" favoriteWidgets" ]; //
 If using post method
```

Example

The form handler deals with these multi-value fields, displaying their values within the Web page.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" >

<head>

<title>Membership Form</title>

</head>

<body>

Membership Form

<p>Thanks for choosing to join The PhoneClub. To register, please fill in
your details below and click Send Details.</p>

           <form action="process_registration_multi.php" method="post" >

             <div style="width: 30em;" >

               <label for="firstName" >First name</label>

               <input type="text" name="firstName" id="firstName" value=""
 />


               <label for="lastName" >Last name</label>

               <input type="text" name="lastName" id="lastName" value="" /
>


               <label for="password1" >Choose a password</label>

               <input type="password" name="password1" id="password1" valu
e="" />

               <label for="password2" >Retype password</label>

               <input type="password" name="password2" id="password2" valu
e="" />


               <label for="genderMale" >Are you male...</label>

               <input type="radio" name="gender" id="genderMale" value="M"
 />

               <label for="genderFemale" >...or female?</label>

               <input type="radio" name="gender" id="genderFemale" value="
F" />


               <label for="favoriteWidgets" >What are your favorite widget
s?</label>
```

```
              <select name="favoriteWidgets[]" id="favoriteWidgets" size=
"3" multiple="multiple" >
                <option value="superWidget" >The SuperWidget</option>
                <option value="megaWidget" >The MegaWidget</option>
                <option value="wonderWidget" >The WonderWidget</option>
              </select>


              <label for="newsletterWidgetTimes" >Do you want to receive
our 'PhoneTimes' newsletter?</label>
              <input type="checkbox" name="newsletter[]" id="newsletterPh
oneTimes" value="widgetTimes" />
              <label for="newsletterFunWithWidgets" >Do you want to recei
ve our 'Fun with Widgets'newsletter?</label>
              <input type="checkbox" name="newsletter[]" id="newsletterFu
nWith Widgets" value="funWithWidgets" />
              <label for="comments" >Any comments?</label>
              <textarea name="comments" id="comments" rows="4" cols="50"
></textarea>


              <div style="clear: both;" >
                <input type="submit" name="submitButton" id="submitButton
" value="Send Details" />
                <input type="reset" name="resetButton" id="resetButton" v
alue="Reset Form" style="margin-right: 20px;" />
              </div>
            </div>
          </form>


        </body>
      </html>
```

Now save the following script as process_registration_multi.php in your document root folder:

```
        <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
 >
        <head>
          <title>Thank You</title>
          </head>
          <body>
            Thank You
            <p>Thank you for registering. Here is the information you submi
tted:</p>
        <?php

        $favoriteWidgets ="" ;
        $newsletters ="" ;

        if (isset($_POST[" favoriteWidgets" ])) {
          foreach ($_POST[" favoriteWidgets" ] as $widget) {
            $favoriteWidgets.= $widget." ," ;
          }
        }

        if (isset($_POST[" newsletter" ])) {
          foreach ($_POST[" newsletter" ] as $newsletter) {
            $newsletters.= $newsletter." ," ;
          }
        }
      $favoriteWidgets = preg_replace(" /, $/" ," " , $favoriteWidgets);
      $newsletters = preg_replace(" /, $/" ," " , $newsletters);
?>
          <dl>
            <dt>First name</dt><dd><?php echo $_POST[" firstName" ]?></dd>
            <dt>Last name</dt><dd><?php echo $_POST[" lastName" ]?></dd>
            <dt>Password</dt><dd><?php echo $_POST[" password1" ]?></dd>
            <dt>Retyped password</dt><dd><?php echo $_POST[" password2" ]?>
</dd>
            <dt>Gender</dt><dd><?php echo $_POST[" gender" ]?></dd>
            <dt>Favorite widgets</dt><dd><?php echo $favoriteWidgets?></dd>
            <dt>You want to receive the following newsletters:</dt><dd>
            <?php echo $newsletters?></dd>
            <dt>Comments</dt><dd><?php echo $_POST[" comments" ]?></dd>
          </dl>


        </body>
      </html>
```

How to make a redirect in PHP?

Redirection from one page to another in PHP is commonly achieved using the following two ways:

Using Header Function in PHP:
The header() function is an inbuilt function in PHP which is used to send the raw HTTP (Hyper Text Transfer Protocol) header to the client.

Syntax:

```
header( $header, $replace, $http_response_code )
```

Parameters: This function accepts three parameters as mentioned above and described below:

- **$header:** This parameter is used to hold the header string.
- **$replace:** This parameter is used to hold the replace parameter which indicates the header should replace a previous similar header, or add a second header of the same type. It is optional parameter.
- **$http_response_code:** This parameter hold the HTTP response code.

Below program illustrates the header() function in PHP:
Program:

```
<?php



// Redirect browser

header("Location: https://www.geeksforgeeks.org");



exit;

?>
```

Note: The die() or exit() function after header is mandatory. If die() or exit() is not put after the header(‘Location: ….’) then script may continue resulting in unexpected behavior. For example, result in content being disclosed that actually wanted to prevent with the redirect (HTTP 301).
Using JavaScript via PHP:
The windows.location object in JavaScript is used to get the current page address(URL) and to redirect the browser to a new page. The window.location object contains the crucial information about a page such as hostname, href, pathname, port etc.
Example:

```
<html>
    <head>
        <title>window.location function</title>
    </head>
    <body>
    <p id="demo"></p>

    <script>
        document.getElementById("demo").innerHTML =
            "URL: " + window.location.href +"</br>";
        document.getElementById("demo").innerHTML =
        document.getElementById("demo").innerHTML +
        "Hostname: " + window.location.hostname + "</br>";
        document.getElementById("demo").innerHTML =
        document.getElementById("demo").innerHTML +
        "Protocol: " + window.location.protocol + "</br>";
    </script>
    </body>
</html>
```

Output:

```
URL: https://ide.geeksforgeeks.org/tryit.php

Hostname: ide.geeksforgeeks.org

Protocol: https:
```

PHP is a server-side scripting language designed specifically for web development.

PHP Sessions

A session is a way to store information (in variables) to be used across multiple pages.Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

A session is started with the `session_start()` function.Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

ExampleGet your own PHP Server

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
```

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

Output:- Session variables are set.

Note: The `session_start()` function must be the very first thing in your document. Before any HTML tags.

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Output:-

```
Favorite color is green.
Favorite animal is cat.
```

Another way to show all the session variable values for a user session is to run the following code:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>
```

Output:- Array ([favcolor] => green [favanimal] => cat)

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Output:- Array ([favcolor] => yellow [favanimal] => cat)

Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

Output:- All session variables are now removed, and the session is destroyed.

Lab Exercises

2. Write a PHP script to calculate the area of a circle using a constant for PI.
3. Create a form with a dropdown menu for selecting a country and display the selected country.
4. Implement a session-based login system with a logout feature.

Conclusion:-

Students will be evaluated based on their ability to write and execute PHP scripts, handle forms, and use sessions effectively.