KISHKINDA UNIVERSITY

ಕಿಷ್ಕಿಂದ ವಿಶ್ವವಿದ್ಯಾಲಯ

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Advancing Knowledge Transforming Live

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

Vision of the University

To develop an ethical and a competent global workforce with an inquisitive mind.

Mission of the University

- To equip the students with professional skills, ethical values and competency.
- To offer state of the art programs by collaborating with industry, academia and society
- To undertake collaborative & inter disciplinary research for overall development.
- To develop leaders and entrepreneurs for the real and sustainable world

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY MANUAL

WEB TECHNOLOGIES LAB

B. Tech 5th Semester

2025-26

Signature of Faculty	**Signature of Chairperson**
1. Mrs. Prathiba Shanbog P S	Dr. Rajashree V Biradar
2. Mr. Imran Ahamed D	

Signature of Instructors

1. Ms. Ashwini M
2. Ms. G Anjanasree

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision of the department

To be a leading center of excellence in computer science education and research, dedicated to producing innovative, ethical, managerial and inquisitive professionals capable of addressing global challenges.

Mission of the department

Mission 1: Offer a cutting-edge curriculum that equips students with the knowledge and skills to excel in the rapidly evolving field of computer science.

Mission 2: Foster a vibrant research environment that encourages collaboration and breakthroughs in technology, driving advancements that address global challenges.

Mission 3: Equip students with a strong ethical and managerial foundation, ensuring they become responsible technologists dedicated to societal and environmental well-being.

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES

After 3-5 years of graduation, the graduates will be able to

PEO1: Graduates will create innovative solutions to global challenges using their computer science expertise.

PEO2: Graduates will lead with integrity, making decisions that benefit society and the environment.

PEO3: Graduates will pursue ongoing learning and research to stay at the forefront of technological advancements.

KISHKINDA UNIVERSITY

(AStatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

Engineering graduates will be able to:

PO1: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Demonstrate knowledge understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

At the end of 4 years, Computer Science and Engineering graduates will be able to:

PSO1: Develop proficiency in applying programming principles, algorithms, and modern software engineering practices to design, develop, and deploy robust software solutions across various platforms.

PSO2: Cultivate skills in employing data analysis techniques, machine learning models, and AI tools to extract insights, drive innovation, and support decision-making in real-world applications.

PSO3: Acquire proficiency to design, implement, and manage complex computer- based systems, ensuring seamless integration of hardware and software components to meet the specified needs.

Course Code	Course Title						Core/Elective
23CSE510	**WEB TECHNOLOGIES**						**CORE**
Prerequisite	Contact Hours Per Week				CIE	SEE	Credits
	L	T	D	P			
–	–	–	–	-	20	00	4

Course Objectives:

This course will enable students to,

- Illustrate the Semantic Structure of HTML and CSS
- Compose forms and tables using HTML and CSS
- Design Client-Side programs using JavaScript and Server-Side programs using PHP
- Infer Object Oriented Programming capabilities of PHP
- Integrating PHP in HTML to access data from backend.

Course Outcomes:

At the end of the course, the student will be able to:

- Acquire the fundamentals of HTML and CSS, including semantic markup and the box model.
- Develop structured tables and forms with accessibility features and implement responsive layouts using CSS frameworks.
- Utilize JavaScript for client-side scripting, DOM manipulation, event handling, and media integration.
- Implement server-side development using PHP, handling user input, file operations.
- Apply OOP Concepts in PHP and integrate MySQL for database management.

KISHKINDA UNIVERSITY

(A StatePrivateUniversityEstablishedbytheKarnatakaActNo.20of2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Subject Name: Web Technologies Lab **Subject Code: 23CSE510** **Semester: 5TH**

CONo.	Course Outcome	Taxonomy Level
23CSE510.1	Acquire the fundamentals of HTML and CSS, including semantic markup and the box model.	Understanding
23CSE510.2	Develop structured tables and forms with accessibility features and implement responsive layouts using CSS frameworks.	Applying
23CSE510.3	Utilize JavaScript for client-side scripting, DOM manipulation, event handling, and media integration.	Applying
23CSE510.4	Implement server-side development using PHP, handling user input, file operations.	Applying
23CSE510.5	Apply OOP Concepts in PHP and integrate MySQL for database management.	Applying

I. Case Study

Perform a case study by Developing the Website for the Employee Management System

II. List of Experiments

1. **Share Your Travel Photos** – A web app for users to upload, view, and organize travel photos.
2. **Book Repository Customer Relations Management** – A CRM system to manage customers and orders for a book repository.
3. **Art Store** – An online art store showcasing artworks with shopping cart and user login features.
4. **Simple Login Form Pre validation** – A login form using JavaScript for client-side input validation.
5. **Write a Node Highlighting Helper Script** – A Java script to traverse and highlight HTML document nodes.
6. **Progressive Enhancement Art Gallery Search** – A search feature for an art gallery that adapts to browser capabilities using progressive enhancement.

Program 1: Share your Travel Photos - A web app for users to upload, view and organize travel photos.

Aim

To design and implement a web application using HTML, CSS, and JavaScript that allows users to upload, view, and organize travel photos.

Objectives

1. To provide a user-friendly interface for uploading and displaying travel photos.
2. To organize photos in a gallery format for easy viewing.
3. To apply HTML for structure, CSS for styling, and JavaScript for dynamic behavior.
4. To simulate a real-world travel photo-sharing platform for users.

Requirements

- Software: Any web browser (Chrome/Firefox/Edge), Text Editor (VS Code/Notepad++).
- Hardware: Standard desktop/laptop with minimum 2 GB RAM.
- Technologies Used:
  - HTML: Provides webpage structure.
  - CSS: Improves the look and feel of the gallery.
  - JavaScript: Handles uploading and organizing photos dynamically.

Problem Statement

Travelers often capture and store photos on different devices, making it difficult to organize and share them. A web-based photo repository is required where users can upload, view, and organize travel photos in a clean gallery view.

Procedure

1. Create a new folder `travel-photos`.
2. Inside the folder, create three files: index.html, style.css, script.js.
3. In index.html, add an upload button and gallery section.
4. Use style.css to design a responsive gallery layout.
5. In script.js, write logic to handle image uploads, preview, and organization.
6. Open index.html in a browser and test by uploading travel photos.

Program Code

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Share Your Travel Photos</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <h1>Share Your Travel Photos</h1>
    <p>Upload, view, and organize your travel memories</p>
  </header>

  <main>
    <section class="upload-section">
      <input type="file" id="photoUpload" accept="image/*" multiple>
      <button onclick="uploadPhotos()">Upload Photos</button>
    </section>

    <section class="gallery-section">
      <h2>Photo Gallery</h2>
      <div id="gallery" class="gallery"></div>
    </section>
  </main>

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

style.css

```
body {
  font-family: Arial, sans-serif;
  background: #f5f5f5;
  margin: 0;
  padding: 0;
}
header {
  background: #2196f3;
  color: white;
  text-align: center;
  padding: 1rem;
}
.upload-section {
  text-align: center;
  margin: 1rem;
}
input[type="file"] {
  margin: 10px;
}
.gallery {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 15px;
  padding: 15px;
}
.photo-card {
  background: white;
  border-radius: 10px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.2);
  padding: 10px;
  text-align: center;
  overflow: hidden;
}
```

```
.photo-card img {
  max-width: 100%;
  border-radius: 8px;
  height: 150px;
  object-fit: cover;
}
.photo-info {
  margin-top: 8px;
}
.photo-info input {
  width: 90%;
  padding: 5px;
  margin: 3px 0;
}
.delete-btn {
  background: crimson;
  color: white;
  border: none;
  padding: 5px 10px;
  border-radius: 5px;
  cursor: pointer;
  margin-top: 5px;
}
.delete-btn:hover {
  background: darkred;
}
```

Script.js

```
let gallery = document.getElementById("gallery");
let photos = [];
function uploadPhotos() {
    let input = document.getElementById("photoUpload");
    let files = input.files;
    for (let file of files) {
        let reader = new FileReader();
```

```
          reader.onload = function(e) {
      let photo = {
        src: e.target.result,
        title: "Untitled",
        album: "General"
      };
      photos.push(photo);
      displayPhotos();
    };
    reader.readAsDataURL(file);
  }
}

function displayPhotos() {
  gallery.innerHTML = "";
  photos.forEach((photo, index) => {
    let card = document.createElement("div");
    card.classList.add("photo-card");

    card.innerHTML = `
      <img src="${photo.src}" alt="Travel Photo">
      <div class="photo-info">
        <input type="text" value="${photo.title}" onchange="updateTitle(${index}, this.value)">
        <input type="text" value="${photo.album}" onchange="updateAlbum(${index}, this.value)">
        <button class="delete-btn" onclick="deletePhoto(${index})">Delete</button>
      </div>
    `;
    gallery.appendChild(card);
  });
}
function updateTitle(index, newTitle) {
  photos[index].title = newTitle;
}
function updateAlbum(index, newAlbum) {
  photos[index].album = newAlbum;
```

```
}
function deletePhoto(index) {
  photos.splice(index, 1);
  displayPhotos();
}
```

Output

User can upload multiple travel photos.

Uploaded photos are displayed in a responsive gallery view.

Photos remain organized in grid format.

Result

The Travel Photo Sharing Web Application was successfully designed and implemented using HTML, CSS and JAVASCRIPT.

Viva Questions

1. What are the advantages of a web-based photo sharing system?
2. How does JavaScript handle file uploads in the browser?
3. Explain the role of CSS Grid in the gallery layout.
4. What is the use of FileReader API in JavaScript?
5. How can we extend this project to include online storage (like cloud or database)?

Program 2: Book Repository Customer Relations Management - A CRM system to manage customers and orders for a book repository.

Aim

To design and implement a CRM system using HTML, CSS, and JavaScript that allows managing customers and book orders for a book repository.

Objectives

1. To provide a simple user interface for managing customers and their orders.
2. To demonstrate the integration of HTML (structure), CSS (design), and JavaScript (logic) in web application development.
3. To maintain relationships between customers and their book orders.
4. To enable adding, listing, and deleting of customers and orders dynamically in a browser.

Requirements

- Software: Any web browser (Chrome/Firefox/Edge), Text Editor (VS Code/Notepad++).
- Hardware: Standard desktop/laptop with minimum 2 GB RAM.
- Technologies Used: HTML Structure, CSS Styling, JavaScript Logic.

Problem Statement

A book repository needs a simple Customer Relationship Management (CRM) System to store customers' details and their corresponding orders. The system should support adding customers, placing book orders, listing them, and removing records dynamically without a database.

Procedure

1. Create a new folder 'book-crm'.
2. Inside the folder, create three files: index.html, style.css, script.js.
3. Write the HTML code to design input forms for adding customers and orders.
4. Apply CSS styles for layout and UI design.
5. Write JavaScript functions to handle adding, displaying, and deleting customers and orders.
6. Open index.html in a browser to test functionality.

Program Code

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Book Repository CRM</title>
  <link rel="stylesheet" href="Book.css">
</head>
<body>
  <header>
    <h1> Book Repository CRM</h1>
    <p>Manage customers & book orders easily</p>
  </header>
  <main>
    <section class="form-section">
      <h2>Add Customer</h2>
      <input type="text" id="customerName" placeholder="Customer Name">
      <input type="email" id="customerEmail" placeholder="Customer Email">
      <button onclick="addCustomer()">Add Customer</button>
    </section>
    <section class="form-section">
      <h2>Add Order</h2>
      <select id="customerSelect"></select>
      <input type="text" id="bookTitle" placeholder="Book Title">
      <input type="number" id="bookQty" placeholder="Quantity" min="1">
      <button onclick="addOrder()">Add Order</button>
    </section>
     <section class="list-section">
      <h2>Customers</h2>
      <ul id="customerList"></ul>
    </section>
    <section class="list-section">
      <h2>Orders</h2>
      <ul id="orderList"></ul>
    </section>
  </main>
 <script src="book.js"></script>
</body>
</html>
```

book.css

```
body {
  font-family: Arial, sans-serif;
  background: #f8f9fa;
  margin: 0;
  padding: 0;
}
header {
  background: #4a90e2;
  color: white;
  text-align: center;
  padding: 1rem;
}
main {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 20px;
  padding: 20px;
}
.form-section, .list-section {
  background: white;
  padding: 15px;
  border-radius: 10px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.2);
}
h2 {
  margin-top: 0;}
```

```css
input, select, button {
  width: 100%;
  padding: 8px;
  margin: 5px 0;
  border: 1px solid #ccc;
  border-radius: 6px;
}
button {
  background: #4a90e2;
  color: white;
  font-weight: bold;
  cursor: pointer;
  border: none;
}
button:hover {
  background: #357abd;
}
ul {
  list-style: none;
  padding: 0;
}
li {
  background: #f1f1f1;
  margin: 5px 0;
  padding: 8px;
  border-radius: 6px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}
.delete-btn {
  background: crimson;
  color: white;
  border: none;
  padding: 5px 8px;
  border-radius: 5px;
  cursor: pointer;
}
.delete-btn:hover {
  background: darkred;
}
```

book.js

```
let customers = [];
let orders = [];

function addCustomer() {
  let name = document.getElementById("customerName").value.trim();
  let email = document.getElementById("customerEmail").value.trim();

  if (!name || !email) {
    alert("Please enter customer details.");
    return;
  }

  let customer = { id: Date.now(), name, email };
  customers.push(customer);
  document.getElementById("customerName").value = "";
  document.getElementById("customerEmail").value = "";
  updateCustomerList();
  updateCustomerSelect();
}
function addOrder() {
  let customerId = document.getElementById("customerSelect").value;
  let bookTitle = document.getElementById("bookTitle").value.trim();
  let qty = document.getElementById("bookQty").value;
  if (!customerId || !bookTitle || qty <= 0) {
    alert("Please fill in all order details.");
    return;  }
  let order = {
    id: Date.now(),
    customerId,
    bookTitle,
    qty
  };
  orders.push(order);
  document.getElementById("bookTitle").value = "";
  document.getElementById("bookQty").value = "";
  updateOrderList();
}

function updateCustomerList() {
  let list = document.getElementById("customerList");
  list.innerHTML = "";
  customers.forEach(c => {
    let li = document.createElement("li");
    li.innerHTML = `${c.name} (${c.email})
      <button class="delete-btn" onclick="deleteCustomer(${c.id})">Delete</button>`;
    list.appendChild(li);
  });
}

function updateOrderList() {
  let list = document.getElementById("orderList");
  list.innerHTML = "";
  orders.forEach(o => {
    let customer = customers.find(c => c.id == o.customerId);
    let li = document.createElement("li");
    li.innerHTML = `${o.bookTitle} (x${o.qty}) - for ${customer ? customer.name : "Unknown"}

      <button class="delete-btn" onclick="deleteOrder(${o.id})">Delete</button>`;
    list.appendChild(li);
  });
}
```

```
function updateCustomerSelect() {
  let select = document.getElementById("customerSelect");
  select.innerHTML = "<option value=''>Select Customer</option>";
  customers.forEach(c => {
    let option = document.createElement("option");
    option.value = c.id;
    option.textContent = c.name;
    select.appendChild(option);
  });
}

function deleteCustomer(id) {
  customers = customers.filter(c => c.id !== id);
  orders = orders.filter(o => o.customerId != id); // remove related orders
  updateCustomerList();
  updateOrderList();
  updateCustomerSelect();
}

function deleteOrder(id) {
  orders = orders.filter(o => o.id !== id);
  updateOrderList();
}
```

Output

Customers can be added with their name and email.

Orders can be added for existing customers.

Lists of customers and orders are displayed dynamically.

Deleting a customer removes all their orders.

Result

The Book Repository CRM was successfully designed and implemented using HTML, CSS, and JavaScript to manage.

Viva Questions

1. What is CRM and why is it important in business?
2. How does JavaScript manipulate the DOM?
3. What is the role of CSS in web application development?
4. Why do we use arrays and objects in this program?
5. How can this system be extended with a database?

Program 3: Art - Store- An online art store showcasing artworks with shopping cart and user login features.

Aim

To design and implement an online art store using HTML, CSS, and JavaScript with features such as user login, product showcase, and a shopping cart system.

Objectives

1. To create a responsive web application for showcasing artworks.
2. To provide login functionality for user authentication.
3. To implement a shopping cart for adding and removing artworks.
4. To apply HTML for structure, CSS for styling, and JavaScript for interactive behavior.

Requirements

- Software: Any modern web browser (Chrome/Firefox/Edge), Text Editor (VS Code/Notepad++).
- Hardware: Standard desktop/laptop with at least 2 GB RAM.
- Technologies Used:
  - HTML: Page structure for artworks and login system.
  - CSS: Styling for store layout, product cards, and forms.
  - JavaScript: Handles login validation and shopping cart functionality.

Problem Statement

Art galleries and independent artists need an online platform to display and sell their artwork. A simple web-based store is required where users can browse artworks, log in, and add items to a shopping cart.

Procedure

1. Create a project folder named `art-store`.
2. Inside the folder, create three files: index.html, style.css, script.js.
3. In index.html, design login form, artwork gallery, and shopping cart sections.
4. Use style.css for layout, colors, and responsive design.
5. In script.js, implement login validation and cart operations (add/remove items).
6. Open index.html in a browser to test functionalities.

Program Code

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Art Store</title>
 <link rel="stylesheet" href="art.css">
</head>
<body>
 <header>
 <h1>Online Art Store</h1>
 <nav>
 <button onclick="showLogin()">Login</button>
 <button onclick="showCart()">Cart (<span id="cart-count">0</span>)</button>
 </nav>
 </header>
 <main>
 <!-- Login Section -->
 <section id="login-section" class="hidden">
 <h2>User Login</h2>
 <form onsubmit="return loginUser()">
 <input type="text" id="username" placeholder="Enter Username" required>
 <input type="password" id="password" placeholder="Enter Password" required>
 <button type="submit">Login</button>
 </form>
 </section>
<!-- Gallery Section -->
 <section id="gallery-section">
 <h2>Artworks</h2>
 <div class="gallery">
 <div class="art-card">
 <img src="https://via.placeholder.com/200" alt="Art 1">
```

```html
 <h3>Abstract Art</h3>
 <p>$50</p>
 <button onclick="addToCart('Abstract Art', 50)">Add to Cart</button>
 </div>
 <div class="art-card">
 <img src="https://via.placeholder.com/200" alt="Art 2">
 <h3>Landscape Painting</h3>
 <p>$70</p>
 <button onclick="addToCart('Landscape Painting', 70)">Add to Cart</button>
 </div>
 </div>
 </section>
 <!-- Cart Section -->
 <section id="cart-section" class="hidden">
 <h2>Your Shopping Cart</h2>
 <ul id="cart-list"></ul>
 <p>Total: $<span id="cart-total">0</span></p>
 <button onclick="checkout()">Checkout</button>
 </section>
 </main>
 <script src="art.js"></script>
</body>
</html>
```

art.css

```css
body {
 font-family: Arial, sans-serif;
 margin: 0;
 padding: 0;
 background: #f9f9f9;
}
header {
 background: #333;
 color: white;
 padding: 15px;
 display: flex;
 justify-content: space-between;
 align-items: center;
}
header h1 {
 margin: 0;
}
nav button {
 margin-left: 10px;
 padding: 8px 12px;
 border: none;
 cursor: pointer;
 background: #ff6f61;
 color: white;
 border-radius: 5px;
}
nav button:hover {
 background: #e65c50;
}
.hidden {
 display: none;
}
.gallery {
 display: flex;
 gap: 20px;
 flex-wrap: wrap;
 padding: 20px;
}
.art-card {
 background: white;
 padding: 15px;
 border-radius: 8px;
 box-shadow: 0 2px 5px rgba(0,0,0,0.1);
 text-align: center;
 width: 200px;
}
```

```
.art-card img {
 max-width: 100%;
 border-radius: 5px;
}
form {
 display: flex;
 flex-direction: column;
 gap: 10px;
 max-width: 300px;
 margin: auto;
 padding: 20px;
}
form input, form button {
 padding: 10px;
 border-radius: 5px;
 border: 1px solid #ccc;
}
form button {
 background: #4caf50;
 color: white;
 border: none;
}
#cart-list {
 list-style: none;
}
#cart-list li {
 padding: 5px;
 border-bottom: 1px solid #ddd;
}
```

art.js

```
let cart = [];
let total = 0;
function showLogin() {
 document.getElementById('login-section').classList.toggle('hidden');
 document.getElementById('cart-section').classList.add('hidden');
}
function loginUser() {
 const username = document.getElementById('username').value;
 const password = document.getElementById('password').value;
 if (username === "user" && password === "pass") {
  alert("Login Successful!");
   document.getElementById('login-section').classList.add('hidden');
 } else {
 alert("Invalid Credentials!");
 }
 return false;
}
function addToCart(item, price) {
 cart.push({item, price});
 total += price;
 updateCart();
}
function updateCart() {
 document.getElementById('cart-count').innerText = cart.length;
 let cartList = document.getElementById('cart-list');
 cartList.innerHTML = "";
 cart.forEach((c, i) => {
 let li = document.createElement('li');
 li.innerText = c.item + " - $" + c.price;
 cartList.appendChild(li);
 });
 document.getElementById('cart-total').innerText = total;
}
function showCart() {
 document.getElementById('cart-section').classList.toggle('hidden');
 document.getElementById('login-section').classList.add('hidden');
}
function checkout() {
 alert("Checkout successful! Total: $" + total);
```

```
cart = [];
total = 0;
updateCart();
 showCart();
}
```

Output

- User can log in with credentials.
- Artworks are displayed in a gallery format.
- Users can add artworks to a shopping cart and view the total.
- Checkout clears the cart.

Result

The Art Store Web Application was successfully designed and implemented using HTML, CSS, and JavaScript with login and shopping cart features.

Viva Questions

1. What is the importance of a shopping cart in an e-commerce website?
2. How does JavaScript manage dynamic elements like adding items to cart?
3. Why is login authentication important in web applications?
4. What are the advantages of separating HTML, CSS, and JavaScript files?
5. How can this project be extended to include backend integration?

Program 4: Simple Login Form Pre validation – A login form using JavaScript for client-side input validation.

Aim:

To design and implement a Login Form with client-side input validation using HTML, CSS, and JavaScript.

Apparatus / Software Requirements:

- Hardware: Computer with minimum 2GB RAM
- Software: Web Browser (Google Chrome, Firefox, Edge), Text Editor (VS Code, Sublime, Notepad++) Languages: HTML, CSS, JavaScript

Theory

Web forms are widely used to collect user data such as login credentials, registrations, etc. Client-side validation is done using JavaScript before sending data to the server. This ensures:

1. Faster response for the user.
2. Prevention of invalid data submission.
3. Enhanced user experience.

In this experiment, a simple login form is created with validation for:
Email must not be empty and should be in valid format.
Password must not be empty and should contain at least 6 characters.

Algorithm / Procedure:

1. Start the program.
2. Create a HTML form with email and password input fields.
3. Add a submit button.
4. Style the form using CSS for better appearance.
5. Write a JavaScript validation function:
6. Check if email field is empty or invalid format.
7. Check if password field is empty or length < 6.
8. Display error messages under respective fields if invalid.
9. If both inputs are valid, display a success message.
10. Stop.

Program:

HTML Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Login Form Validation</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<div class="login-container">
<h2>Login</h2>
<form id="loginForm">
<div class="form-group">
<label for="email">Email</label>
<input type="text" id="email" placeholder="Enter email">
<div id="emailError" class="error"></div>
</div>

<div class="form-group">

<label for="password">Password</label>
<input type="password" id="password" placeholder="Enter password">
<div id="passwordError" class="error"></div>
</div>

<button type="submit">Login</button>
</form>
</div>

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

CSS Code:

```
body {
font-family: Arial, sans-serif;
background: #f5f5f5;
display: flex;
justify-content: center;
 align-items: center;
height: 100vh;
}

.login-container {
background: #fff;
padding: 25px;
border-radius: 12px;
box-shadow: 0px 4px 8px rgba(0,0,0,0.1);
width: 300px;
}

h2 {
text-align: center;
margin-bottom: 20px;
color: #333;
}

.form-group {
margin-bottom: 15px;
}

label {
font-size: 14px;
display: block;
margin-bottom: 5px;
color: #444;
}
```

```
input {
width: 100%;
padding: 10px;
border: 1px solid #ccc;
border-radius: 8px;
font-size: 14px;
}

.error {
color: red;
font-size: 12px;
margin-top: 5px;
}

button {
width: 100%;
padding: 10px;
background: #4CAF50;
border: none;
color: white;
font-size: 16px;
border-radius: 8px;
cursor: pointer;
}

button:hover {
background: #45a049;
}
```

JavaScript Code:

```
document.getElementById("loginForm").addEventListener("submit", function(e) { e.preventDefault(); //
prevent form submit

let email = document.getElementById("email").value.trim();
let password = document.getElementById("password").value.trim();
let emailError = document.getElementById("emailError");
let passwordError = document.getElementById("passwordError");

// Clear errors emailError.textContent = ""; passwordError.textContent = "";

let isValid = true;

// Email validation if (email === "") {
emailError.textContent = "Email is required"; isValid = false;
} else if (!/^\S+@\S+\.\S+$/.test(email)) { emailError.textContent = "Invalid email format"; isValid =
false;
}

// Password validation if (password === "") {
passwordError.textContent = "Password is required"; isValid = false;
} else if (password.length < 6) {
passwordError.textContent = "Password must be at least 6 characters"; isValid = false;
}

if (isValid) {
alert("Login Successful!");
// Form can be submitted or redirected here
}
});
```

Output:

If fields are empty, Shows error messages.
If email format is invalid, Shows 'Invalid email format'.
If password is less than 6 characters, shows 'Password must be at least 6 characters'.
If all inputs are valid, Shows 'Login Successful!'

Result:

The login form with JavaScript pre-validation is successfully implemented.

Viva Questions:

1. What is the difference between client-side and server-side validation?
2. Why do we use JavaScript for form validation?
3. What is the role of regular expressions in validation?
4. Can client-side validation alone be trusted for security? Why?
5. What are HTML5 input validation attributes (like required, pattern)?

Program 5: Write a Node Highlighting Helper Script – A Java script to traverse and highlight HTML document nodes.

Aim:

To design and implement a Node Highlighting Helper Script that traverses the HTML document nodes and highlights them dynamically using JavaScript.

Apparatus / Software Requirements:

- Hardware: Computer with minimum 2GB RAM
- Software: Web Browser (Google Chrome, Firefox, Edge), Text Editor (VS Code, Sublime, Notepad++)
- Languages: HTML, CSS, JavaScript

Theory:

In the Document Object Model (DOM), HTML elements are treated as nodes. Traversing the DOM allows developers to access and manipulate elements dynamically.

The Node Highlighting Helper Script:

Iterates over nodes in the document.

Highlights them visually when traversed.

Helps in debugging, learning DOM structure, and visualizing parent-child relationships.

Key Concepts Used:

document.body - to access the main document body.

Recursion - to traverse nodes hierarchically.

style manipulation - to apply background color highlighting.

Algorithm / Procedure:

1. Start the program.
2. Create a HTML document with nested elements (paragraphs, divs, headings).
3. Define a CSS style for highlighted elements.
4. Write a JavaScript recursive function to:
5. Traverse each child node.
6. Apply highlight effect to the current node.
7. Move to the next node after a delay.
8. Trigger traversal on a button click.
9. End.

Program:

HTML Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Node Highlighting Helper Script</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<h1>Welcome to Node Highlighting</h1>
<p>This is a simple example to demonstrate DOM node traversal and highlighting.</p>

<div>
<h2>Section 1</h2>
<p>First paragraph inside Section 1</p>
</div>

<div>
<h2>Section 2</h2>
<p>First paragraph inside Section 2</p>
<p>Second paragraph inside Section 2</p>
</div>

<button id="startBtn">Start Traversal</button>

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

CSS Code:

```
body {
font-family: Arial, sans-serif; margin: 20px;
}
```

```
h1 {
color: #333;
}
button {
margin-top: 20px; padding: 10px 15px; font-size: 16px; background: #4CAF50; color: white; border:
none;
border-radius: 8px; cursor: pointer;
}

button:hover { background: #45a049;
}
.highlight {
background-color: yellow; border: 1px solid orange;
}
```

JavaScript Code:

```
// Recursive function to traverse nodes function traverseNodes(node, callback) {
if (node.nodeType === 1) { // Only element nodes callback(node);
let children = node.childNodes;
for (let i = 0; i < children.length; i++) { traverseNodes(children[i], callback);
}
}
}

// Highlighting function function highlightNode(node) {
node.classList.add("highlight"); setTimeout(() => {
node.classList.remove("highlight");
}, 800); // Highlight effect for 800ms
}

// Button click event document.getElementById("startBtn").addEventListener("click", function () {
let allNodes = [];

// Collect nodes
traverseNodes(document.body, function (node) { allNodes.push(node);
});
// Sequential highlighting with delay let i = 0;
function highlightNext() { if (i < allNodes.length) {
highlightNode(allNodes[i]); i++;
setTimeout(highlightNext, 1000); // Move to next node after 1 sec
}
}
highlightNext();
});
```

Output:

The page contains some nested HTML elements.
When the user clicks Start Traversal, each HTML element (node) is highlighted one by one in yellow for a short duration.

Result:

The Node Highlighting Helper Script was successfully implemented. The program traverses the HTML document nodes and highlights them sequentially.

Viva Questions:

1. What is the DOM in JavaScript?
2. Differentiate between Node and Element in the DOM.
3. Why do we use recursion in DOM traversal?
4. What are the different node types in the DOM?
5. Can you highlight only text nodes instead of elements? How

Program 6: Progressive Enhancement Art Gallery Search – A search feature for an art gallery that adapts to browser capabilities using progressive enhancement.

Aim:

To develop a search feature for an art gallery that adapts to browser capabilities using Progressive Enhancement with HTML, CSS, and JavaScript.

Requirements:

- A computer with any web browser (Chrome, Firefox, Edge).
- Code Editor (VS Code, Sublime, Notepad++).
- HTML, CSS, and JavaScript knowledge.

Theory:

Progressive Enhancement is a web design strategy that emphasizes starting with a basic, functional experience that works across all browsers and then adding advanced features for browsers that support them.

In this experiment, we build:

A simple HTML search form to filter artworks.
CSS styling to improve user experience.
JavaScript enhancement to enable instant search results without reloading the page.

This ensures the application works on older browsers (basic form submission) and newer ones (dynamic filtering).

Algorithm:

1. Create a basic HTML structure with an input box and a list of artworks.
2. Add CSS styling for layout and presentation.
3. Implement JavaScript to:
4. Capture user input.
5. Filter artworks dynamically as the user types.
6. If JavaScript is disabled, the user can still perform a basic form submission.

Program:

HTML (progressive.html)

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Art Gallery Search</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<h1>Art Gallery</h1>
<!-- Basic Search Form (works without JS) -->
<form id="searchForm">
<label for="search">Search Artworks:</label>
<input type="text" id="search" name="search" placeholder="Type to search...">
<button type="submit">Search</button>
</form>
<ul id="gallery">
<li data-title="Mona Lisa">Mona Lisa</li>
<li data-title="Starry Night">Starry Night</li>
<li data-title="The Scream">The Scream</li>
<li data-title="The Last Supper">The Last Supper</li>
<li data-title="Girl with a Pearl Earring">Girl with a Pearl Earring</li>
</ul>

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

CSS (style.css):

```css
body {
font-family: Arial, sans-serif;
margin: 20px;
background: #f8f8f8;
```

```
color: #333;
}
h1 {
text-align: center;
margin-bottom: 20px;
}

form {
text-align: center;
margin-bottom: 20px;
}

input {
 padding: 8px;
width: 200px;
border: 1px solid #ccc;
border-radius: 5px;
}

button {
padding: 8px 12px;
margin-left: 5px;
background: #4CAF50;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
}
button:hover {
background: #45a049;
}
#gallery {
list-style: none;
padding: 0;
max-width: 400px;
```

```
margin: auto;
}

#gallery li {
background: #fff;
margin: 5px 0;
padding: 10px;
border-radius: 8px;
box-shadow: 0px 2px 5px rgba(0,0,0,0.1);
}
```

JavaScript (script.js):

```
document.addEventListener("DOMContentLoaded", function() { const searchInput =
document.getElementById("search");
const galleryItems = document.querySelectorAll("#gallery li");

searchInput.addEventListener("input", function() { const filter = searchInput.value.toLowerCase();

galleryItems.forEach(item => {
const title = item.dataset.title.toLowerCase(); if (title.includes(filter)) {
item.style.display = "";
} else {
item.style.display = "none";
}
});
});
});
```

Output:

Initially displays all artworks.
When the user types into the search bar, artworks are filtered instantly.
If JavaScript is disabled, form submission still works.

Result:

Thus, a Progressive Enhancement-based Art Gallery Search feature was successfully implemented using HTML, CSS, and JavaScript.

Viva Questions:

1. What is Progressive Enhancement in web development?
2. How does it differ from Graceful Degradation?
3. Why is it important to build a fallback option in web applications?
4. How does JavaScript enhance user experience in this project?
5. Can this technique be applied to mobile-first design?