Lab Manual:

Internet and Web Programming Lab

(17YCS511)

SOCSE

Sandip University Nashik,India

Year: Third Year **Semester – V**
Course: Internet and Web Programming Lab **Course Code: 17YCS511**

Teaching Scheme (Hrs. /Week)				Continuous Internal Assessment (CIA)					End Semester Examination		Total
L	T	P	C	CIA-1	CIA-2	CIA-3	CIA-4	Lab	Theory	Lab	
0	0	4	2	--	--	--	--	25	0	25	50
Max. Time, End Semester Exam (Theory) -00 Hrs.									End Semester Exam (Lab) – 03 Hrs.		

Prerequisites:
C C++, Basics of Computer Networks and Data Communications

Objectives:	
Students are able to:-	
1	Gain comprehensive knowledge of HTML, CSS, and JavaScript, and their role in creating dynamic, responsive web pages.
2	Learn server-side scripting languages like PHP or Node.js, focusing on building robust, secure back-end functionality for web applications.
3	Master the integration of databases using SQL with web applications, emphasizing data retrieval, manipulation, and storage techniques.
4	Explore the development and consumption of Web APIs, understanding RESTful services, and the role of JSON/XML in data interchange.
5	Develop the skills to create full-stack web applications, combining front-end and back-end technologies to build complete, functional web solutions.

Guidelines for Assessment

Term Work assessment shall be conducted for the Project, Tutorials and Seminar. Term work is continuous assessment based on work done, submission of work in the form of report/journal, timely completion, attendance, and understanding. It should be assessed by subject teacher of the institute. At the end of the semester, the final grade for a Term Work shall be assigned based on the performance of the student and is to be submitted to the University.
The experiments from the regular practical syllabus will be performed (15 Marks).
The regular attendance of students during the syllabus practical course will be monitored and marks will be given accordingly (5 Marks).
Good Laboratory Practices (5 Marks)

Guidelines for Laboratory Conduction

Practical/Oral/Presentation shall be conducted and assessed jointly by internal and external examiners. The performance in the Practical/Oral/Presentation examination shall be assessed by at least a pair of examiners appointed as examiners by the University. The examiners will prepare the mark/grade sheet in the format as specified by the University, authenticate and seal it. Sealed envelope shall be submitted to the head of the department or authorized person.

One experiment from the regular practical syllabus will be conducted. (Total 15 Marks).

Complete laboratory journal/records (05 Marks).
Viva-voce (05 Marks).

Operating System recommended :
Programming tools recommended:

Suggested List of Laboratory Assignments

Group A (Compulsory Assignments)

1. Design a web page using different text formatting tags.
2. Design a web page with links to different pages and allow navigation between pages.
3. Design a web page with Image maps.
4. Design a web page with different tables. Design a webpage using table so that the content appears well placed.
5. Design a webpage using frames.
6. Design a web page with a form that uses all types of controls.
7. Design a website using style sheets so that the pages have uniform style.
8. Using Java Script design a web page that prints factorial / Fibonacci series / any given series.
9. Design a form with a test box and a command button. Using JavaScript write a program whether the number entered in the text box is a prime number or not.
10. Design a form and validate all the controls placed on the form using Java Script.
11. Design a DTD, corresponding XML document and display it in browser using CSS.
12. Design an XML document and display it in browser using XSL.
13. Design XML Schema and corresponding XML document.
14 Write a Basic PHP program to display “Hello World” using PHP programming.

Experiments 1:

1. Design a web page using different text formatting tags.

```
<!DOCTYPE html>
<html>
<head>
<title>Text Formatting Demo</title>
</head>
<body>

<h1>Text Formatting Demo</h1>

<h2>Headings</h2>
<p>This is a paragraph of text.</p>
<h3>Subheading</h3>
<p>Another paragraph of text.</p>

<h2>Emphasis</h2>
<p>This text is <b>bold</b>.</p>
<p>This text is <strong>strongly emphasized</strong>.</p>
<p>This text is <i>italic</i>.</p>
<p>This text is <em>emphasized</em>.</p>
<p>This text is <mark>marked</mark>.</p>
<p>This text is <small>small</small>.</p>
<p>This text is <del>deleted</del>.</p>
<p>This text is <sub>subscript</sub>.</p>
<p>This text is <sup>superscript</sup>.</p>

<h2>Other Formatting</h2>
<p>This is a <q>short quotation</q>.</p>
<p>This is a <blockquote>long quotation</blockquote>.</p>
<p>This is <code>code</code>.</p>
```

```
<p>This is <kbd>keyboard input</kbd>.</p>
<p>This is <samp>sample output</samp>.</p>
<p>This is <var>a variable</var>.</p>

</body>
</html>
```

Explanation of Tags:

- <h1> to <h6>: Headings of decreasing size.
- <p>: Paragraphs of text.
- <b>: Bold text.
- : Strongly emphasized text (often rendered bold).
- <i>: Italic text.
- : Emphasized text (often rendered italic).
- <mark>: Marked text (often rendered with a background color).
- <small>: Smaller text.
- <del>: Deleted text (often rendered with a strikethrough).
- <sub>: Subscript text.
- <sup>: Superscript text.
- <q>: Short quotation.
- <blockquote>: Long quotation (often indented).
- <code>: Code (often rendered in a monospace font).
- <kbd>: Keyboard input (often rendered in a monospace font).
- <samp>: Sample output (often rendered in a monospace font).
- <var>: Variable (often rendered in a monospace font).

Experiments 2:

2. Design a web page with links to different pages and allow navigation between pages.

```
<!DOCTYPE html>
<html>
<head>
<title>Navigation Demo</title>
</head>
<body>

<h1>Navigation Demo</h1>

<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="about.html">About</a></li>
    <li><a href="contact.html">Contact</a></li>
  </ul>
</nav>

<h2>Welcome to the Home Page</h2>
<p>This is the content of the home page.</p>

</body>
</html>
```

Explanation:

<nav> element: This semantic tag is used to define a navigation section on your web page.

 element: An unordered list is used to contain the navigation links.

 elements: Each list item represents a single navigation link.

<a> elements: The anchor tags create the actual links.

href attribute: Specifies the destination URL for the link. You'll need to create the about.html and contact.html files to make these links functional.

Text content: The visible text of the link.

Key points:

File structure: Make sure you have the index.html, about.html, and contact.html files in the same directory for the navigation to work correctly.

Relative paths: In this example, we're using relative paths for the href attribute, assuming all the files are in the same folder. If you have a more complex directory structure, you might need to adjust the paths accordingly.

Styling: You can use CSS to style the navigation bar and links to match your website's design.

Active link: You can use JavaScript or CSS to highlight the currently active link, providing visual feedback to the user.

Experiments 3:

3. Design a web page with Image maps.

```
<!DOCTYPE html>
<html>
<head>
<title>Image Map Demo</title>
</head>
<body>

<h1>Image Map Demo</h1>

<img src="map.jpg" alt="Map" usemap="#map">

<map name="map">
  <area shape="rect" coords="0,0,100,100" href="location1.html" alt="Location 1">
  <area shape="circle" coords="200,200,50" href="location2.html" alt="Location 2">
  <area shape="poly" coords="300,0,350,50,250,100" href="location3.html" alt="Location 3">
</map>
```

</body>
</html>
Explanation:

<img> element:
src: Specifies the image file to be used (replace map.jpg with your actual image).
alt: Provides alternative text for screen readers or if the image fails to load.
usemap: Links the image to the map defined below using the #map identifier.

<map> element:
name: Must match the value used in the usemap attribute of the <img> tag.
Contains <area> elements defining the clickable regions on the image.

<area> elements:
shape: Specifies the shape of the clickable area (rect, circle, poly).
coords: Defines the coordinates for the shape:
rect: x1, y1, x2, y2 (top-left and bottom-right corners)
circle: center_x, center_y, radius
poly: x1, y1, x2, y2, ... (vertices of the polygon)
href: Specifies the destination URL for the link when that area is clicked.
alt: Provides alternative text for the area, similar to the <img> tag's alt attribute.

Key points:

Image preparation: You'll need an image file (map.jpg in this example) to use with the image map.

Coordinate accuracy: Make sure the coordinates in the <area> elements accurately define the clickable regions on your image. You might need to use an image editor or online tool to get the exact coordinates.

Fallback links: Consider providing regular text links below the image map for users who have JavaScript disabled or are using assistive technologies that might not fully support image maps.

Accessibility: Ensure the alt attributes for the <img> and <area> elements provide meaningful descriptions for users who cannot see the image.

Experiments 4:

4. Design a web page with different tables. Design a webpage using table so that the content appears well placed.

```
<!DOCTYPE html>
<html>
<head>
<title>Table Layout Demo</title>
<style>
table {
   width: 100%;
   border-collapse: collapse;
}

th, td {
   border: 1px solid black;
   padding: 8px;
   text-align: left;
}

th {
   background-color: #f2f2f2;
}
</style>
</head>
<body>

<h1>Table Layout Demo</h1>

<table>
   <tr>
      <th>Name</th>
      <th>Age</th>
      <th>City</th>
```

```
    </tr>
    <tr>
        <td>John Doe</td>
        <td>30</td>
        <td>New York</td>
    </tr>
    <tr>
        <td>Jane Smith</td>
        <td>25</td>
        <td>London</td>
    </tr>
</table>

<h2>Nested Table</h2>

<table>
    <tr>
        <th>Product</th>
        <th>Description</th>
        <th>Price</th>
    </tr>
    <tr>
        <td>Laptop</td>
        <td>
            <table>
                <tr>
                    <td>Brand:</td>
                    <td>XYZ</td>
                </tr>
                <tr>
                    <td>Model:</td>
                    <td>ABC</td>
                </tr>
            </table>
```

```
        </td>
        <td>$999</td>
    </tr>
</table>

</body>
</html>
```

Explanation:

Basic table:

<table>: Defines the table structure.

<tr>: Represents a table row.

<th>: Defines a table header cell.

<td>: Defines a table data cell.

Nested table:

A table is embedded within a <td> cell of the outer table to create a more complex layout.

CSS styling:

width: 100%: Makes the table take up the full width of its container.

border-collapse: collapse;: Collapses the borders of adjacent cells into a single border.

border: 1px solid black;: Adds a 1-pixel black border to all cells.

padding: 8px;: Adds padding around the content of each cell.

text-align: left;: Aligns the text within each cell to the left.

th { background-color: #f2f2f2; }: Sets a light gray background color for header cells.

Key points:

Semantic structure: Use tables for tabular data, not for general layout purposes. Consider using CSS Grid or Flexbox for more flexible and accessible layouts.

Accessibility: Provide appropriate <caption>, <thead>, <tbody>, and <tfoot> elements to enhance accessibility for screen reader users.

Responsiveness: Use CSS media queries to adjust the table layout for different screen sizes.

Experiments 5:

5. Design a webpage using frames.

```
<!DOCTYPE html>
<html>
<head>
<title>Frames Demo</title>
</head>
<frameset cols="25%,*,25%">
  <frame src="left.html">
  <frame src="main.html" name="main">
  <frame src="right.html">
</frameset>
</html>
```

Explanation:

<frameset> element:

Replaces the <body> tag in a frames-based layout.

cols: Specifies the column widths using percentages or fixed pixel values. In this example:

First column: 25% of the browser width

Second column: Takes up the remaining space (*)

Third column: 25% of the browser width

<frame> elements:

Define the individual frames within the frameset.

src: Specifies the HTML file to be displayed in the frame. You'll need to create left.html, main.html, and right.html files to make this work.

name (optional): Gives the frame a name so you can target it with links from other frames.

Key points:

File structure: Make sure you have the left.html, main.html, and right.html files in the same directory as this main HTML file.

Linking between frames:

To create a link in one frame that targets another frame, use the target attribute in the <a> tag:

HTML

```
<a href="some_page.html" target="main">Link to main frame</a>
```

This will load some_page.html into the frame named "main".

Important considerations:

Accessibility: Frames can create significant accessibility challenges for users with screen readers or other assistive technologies. It's generally recommended to avoid frames whenever possible.

SEO: Search engines might have difficulty indexing content within frames, potentially impacting your website's visibility.

Usability: Frames can lead to a confusing user experience, especially on smaller screens or mobile devices.

Alternatives to frames:

CSS layout techniques: Use CSS Grid or Flexbox to create complex layouts without relying on frames.

IFrames: Consider using <iframe> elements to embed external content within your page, but be mindful of their potential impact on performance and accessibility.

Experiments 6:

6. Design a web page with a form that uses all types of controls.

```
<!DOCTYPE html>
<html>
<head>
<title>Form Controls Demo</title>
</head>
<body>

<h1>Form Controls Demo</h1>

<form>
  <fieldset>
    <legend>Personal Information</legend>

    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>

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

    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required><br><br>

    <label for="dob">Date of Birth:</label>
    <input type="date" id="dob" name="dob"><br><br>

    <label>Gender:</label>
    <input type="radio" id="male" name="gender" value="male">
    <label for="male">Male</label>
    <input type="radio" id="female" name="gender" value="female">
    <label for="female">Female</label><br><br>
```

```
        <label for="country">Country:</label>
        <select id="country" name="country">
            <option value="india">India</option>
            <option value="usa">USA</option>
            <option value="uk">UK</option>
        </select><br><br>

        <label for="bio">Bio:</label>
        <textarea id="bio" name="bio" rows="4" cols="50"></textarea><br><br>
    </fieldset>

    <fieldset>
        <legend>Skills</legend>

        <label for="html">HTML:</label>
        <input type="checkbox" id="html" name="skills" value="html"><br>

        <label for="css">CSS:</label>
        <input type="checkbox" id="css" name="skills" value="css"><br>

        <label for="javascript">JavaScript:</label>
        <input type="checkbox" id="javascript" name="skills" value="javascript"><br><br>
    </fieldset>

    <input type="submit" value="Submit">
    <input type="reset" value="Reset">
</form>

</body>
</html>
```

Explanation of Controls:

- `<input type="text">`: Single-line text input.
- `<input type="email">`: Email input with validation.

<input type="password">: Password input with hidden characters.

<input type="date">: Date picker control.

<input type="radio">: Radio buttons for single-choice selection.

<select> and <option>: Dropdown list for selection.

<textarea>: Multi-line text input.

<input type="checkbox">: Checkboxes for multiple selections.

<input type="submit">: Submit button to send form data.

<input type="reset">: Reset button to clear form fields.

<fieldset> and <legend>: Group related form elements with a caption.

<label> and for attribute: Associate labels with input fields for better accessibility.

Key points:

Form handling: You'll need to add server-side code (e.g., PHP, Python, Node.js) or client-side JavaScript to handle the form submission and process the data.

Validation: Consider adding validation to ensure users enter valid data (e.g., required fields, email format, password strength).

Accessibility: Use semantic HTML and ARIA attributes to make the form accessible to users with disabilities.

Styling: Use CSS to style the form and its elements to match your website's design.

Experiments 7:

7. Design a website using style sheets so that the pages have uniform style.

/* style.css */

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f0f0f0;
    color: #333;
}
```

```
header {
    background-color: #333;
    color: #fff;
    padding: 20px;
    text-align: center;
}

nav ul {
    list-style: none;
    margin: 0;
    padding: 0;
    background-color: #eee;
}

nav li {
    display: inline;
    margin-right: 20px;
}

nav a {
    text-decoration: none;
    color: #333;
}

main {
    padding: 20px;
}

footer {
    background-color: #333;
    color: #fff;
    padding: 10px;
    text-align: center;
```

```
    position: fixed;
    bottom: 0;
    width: 100%;
}
```

Create your HTML pages (e.g., index.html, about.html, contact.html)

```
<!DOCTYPE html>
<html>
<head>
<title>Home Page</title>
<link rel="stylesheet" href="style.css">
</head>
<body>

<header>
    <h1>My Website</h1>
</header>

<nav>
    <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="about.html">About</a></li>
        <li><a href="contact.html">Contact</a></li>
    </ul>
</nav>

<main>
    <h2>Welcome to the Home Page</h2>
    <p>This is the content of the home page.</p>
</main>

<footer>
    <p>&copy; 2023 My Website</p>
</footer>
```

</body>
</html>
Explanation:

External CSS file (style.css):

Contains all the CSS rules that define the visual appearance of your website.

You can reuse this file across multiple HTML pages to maintain a consistent style.

HTML pages:

Include the <link rel="stylesheet" href="style.css"> tag in the <head> section to link the external CSS file.

Use semantic HTML elements (header, nav, main, footer) to structure your content.

The CSS rules in style.css will be applied to these elements, providing a uniform look and feel across all pages.

Key points:

Consistent styling: By using an external CSS file, you can easily update the style of your entire website by making changes in a single location.

Maintainability: This approach makes your code more organized and easier to maintain, especially for larger websites.

Performance: The browser can cache the CSS file, improving page load times for subsequent visits.

Experiments 8:

8. Using Java Script design a web page that prints factorial / Fibonacci series / any given series.

```
<!DOCTYPE html>
<html>
<head>
<title>Series Calculator</title>
</head>
<body>

<h1>Series Calculator</h1>

<label for="seriesType">Select Series Type:</label>
<select id="seriesType">
    <option value="factorial">Factorial</option>
    <option value="fibonacci">Fibonacci</option>
    <option value="custom">Custom</option>
</select><br><br>

<label for="inputNumber">Enter Number:</label>
<input type="number" id="inputNumber"><br><br>

<button onclick="calculateSeries()">Calculate</button>

<p id="result"></p>

<script>
function calculateSeries() {
    const seriesType = document.getElementById("seriesType").value;
    const inputNumber = parseInt(document.getElementById("inputNumber").value);
    let result = "";

    if (seriesType === "factorial") {
        result = calculateFactorial(inputNumber);
    } else if (seriesType === "fibonacci") {
```

```
        result = calculateFibonacci(inputNumber);
    } else if (seriesType === "custom") {
        // You'll need to implement the logic for custom series here
        result = "Custom series calculation not yet implemented.";
    }

    document.getElementById("result").textContent = result;
}

function calculateFactorial(n) {
    if (n === 0 || n === 1) {
        return 1;
    } else {
        return n * calculateFactorial(n - 1);
    }
}

function calculateFibonacci(n) {
    const fibSeries = [0, 1];
    for (let i = 2; i <= n; i++) {
        fibSeries[i] = fibSeries[i - 1] + fibSeries[i - 2];
    }
    return fibSeries.join(", ");
}
</script>

</body>
</html>
```

Explanation:

- HTML structure:
  - Dropdown (<select>) to choose the series type.
  - Number input (<input type="number">) to get the input value.
  - Button (<button>) to trigger the calculation.

Paragraph (<p>) to display the result.

JavaScript functions:

calculateSeries():

Gets the selected series type and input number.

Calls the appropriate calculation function based on the series type.

Updates the result paragraph with the calculated series.

calculateFactorial(n):

Recursively calculates the factorial of n.

calculateFibonacci(n):

Iteratively calculates the Fibonacci series up to the nth term.

Custom series:

The custom option is included, but you'll need to implement the logic for calculating custom series based on your specific requirements.

Key points:

Error handling: You might want to add error handling to check for invalid input (e.g., negative numbers, non-numeric input) and display appropriate messages to the user.

Custom series: The calculateCustomSeries() function is a placeholder. You'll need to define the logic for calculating your desired custom series.

Optimization: For large factorials or Fibonacci numbers, consider using memoization or other optimization techniques to improve performance.

Experiments 9:

9. Design a form with a test box and a command button. Using JavaScript write a program whether the number entered in the text box is a prime number or not.

```
<!DOCTYPE html>
<html>
<head>
<title>Prime Number Checker</title>
</head>
<body>

<h1>Prime Number Checker</h1>

<label for="numberInput">Enter a number:</label>
<input type="number" id="numberInput"><br><br>

<button onclick="checkPrime()">Check</button>

<p id="result"></p>

<script>
function checkPrime() {
    const number = parseInt(document.getElementById("numberInput").value);
    const isPrime = isPrimeNumber(number);

    if (isPrime) {
        document.getElementById("result").textContent = number + " is a prime number.";
    } else {
        document.getElementById("result").textContent = number + " is not a prime number.";
    }
}

function isPrimeNumber(num) {
    if (num <= 1) {
        return false;
```

```
    }
    for (let i = 2; i <= Math.sqrt(num); i++) {
        if (num % i === 0) {
            return false;
        }
    }
    return true;
}
</script>

</body>
</html>
```

Explanation:

HTML structure:

- Input field (<input type="number">) to get the number from the user.
- Button (<button>) to trigger the prime number check.
- Paragraph (<p>) to display the result.

JavaScript functions:

- checkPrime():
  - Gets the number from the input field.
  - Calls the isPrimeNumber() function to check if it's prime.
  - Updates the result paragraph with the appropriate message.
- isPrimeNumber(num):
  - Checks if the given number num is prime:
    - Returns false if num is less than or equal to 1.
    - Iterates from 2 up to the square root of num.
    - If num is divisible by any number in this range, it's not prime, so returns false.
    - If the loop completes without finding any divisors, num is prime, so returns true.

Key points:

Input validation: You might want to add input validation to ensure the user enters a valid positive integer.

Efficiency: The isPrimeNumber function uses the square root optimization to improve performance for larger numbers.

Error handling: Consider adding error handling to gracefully handle cases where the user enters invalid input.

Experiments 10:

10. Design a form and validate all the controls placed on the form using Java Script.

```
<!DOCTYPE html>
<html>
<head>
<title>Form Validation Demo</title>
<style>
.error {
    color: red;
}
</style>
</head>
<body>

<h1>Form Validation Demo</h1>

<form id="myForm" onsubmit="return validateForm()">

    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>

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

    <label for="password">Password:</label>
```

```
    <input type="password" id="password" name="password" required><br><br>

    <label for="confirmPassword">Confirm Password:</label>
    <input type="password" id="confirmPassword" name="confirmPassword" required><br><br>

    <label for="age">Age:</label>
    <input type="number" id="age" name="age" min="18" max="120" required><br><br>

    <input type="submit" value="Submit">
</form>

<p id="errorMessages"></p>

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

    // Name validation
    const name = document.getElementById("name").value;
    if (name.trim() === "") {
        errorMessages += "Name is required.<br>";
        isValid = false;
    }

    // Email validation
    const email = document.getElementById("email").value;
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
        errorMessages += "Invalid email format.<br>";
        isValid = false;
    }
```

```
    // Password validation
    const password = document.getElementById("password").value;
    const confirmPassword = document.getElementById("confirmPassword").value;
    if (password !== confirmPassword) {
        errorMessages += "Passwords do not match.<br>";
        isValid = false;
    }

    // Age validation (handled by HTML5 attributes)

    // Display error messages
    document.getElementById("errorMessages").innerHTML = errorMessages;

    return isValid; // Prevent form submission if not valid
}
</script>

</body>
</html>
```

Explanation:

HTML Form:

Includes input fields for name, email, password, confirm password, and age.

The required attribute ensures that these fields cannot be left blank.

The min and max attributes on the age input restrict the valid age range.

An empty paragraph ( ) is added to display error messages.

JavaScript Validation (validateForm()):

This function is called when the form is submitted (onsubmit event).

It checks each input field for validity and constructs an error message if any validation fails.

The isValid variable tracks the overall validity of the form.

If any validation fails, the error messages are displayed in the errorMessages paragraph, and the form submission is prevented by returning false.

If all validations pass, the form will be submitted as usual.

Key points:

Client-side validation: This JavaScript code performs validation on the user's browser before the form is submitted to the server.

Server-side validation: It's crucial to also implement server-side validation to ensure data integrity and security, as client-side validation can be bypassed.

Error handling: The errorMessages paragraph provides feedback to the user about any validation errors. You can further enhance this with visual cues (e.g., highlighting invalid fields).

Accessibility: Ensure your form is accessible to all users by using semantic HTML, ARIA attributes, and clear labels.

Experiments 11:

11. Design a DTD, corresponding XML document and display it in browser using CSS.

1. DTD (Document Type Definition)

Let's create a DTD for a simple book catalog:

```
<!ELEMENT catalog (book*)>
<!ELEMENT book (title, author, year, price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

This DTD defines a catalog that can contain zero or more book elements. Each book has a title, author, year, and price, all of which contain simple text data.

2. XML Document

Here's an XML document adhering to the above DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "books.dtd">
```

```
<catalog>
   <book>
      <title>The Hitchhiker's Guide to the Galaxy</title>
      <author>Douglas Adams</author>
      <year>1979</year>
      <price>10.99</price>
   </book>
   <book>
      <title>1984</title>
      <author>George Orwell</author>
      <year>1949</year>
      <price>8.50</price>
   </book>
</catalog>
```

3. CSS Styling

Now, let's style this XML using CSS. You'll typically use XSLT (Extensible Stylesheet Language Transformations) to transform your XML into HTML, and then apply CSS to the resulting HTML. However, some modern browsers might have limited support for directly styling XML with CSS.

Here's a simple CSS example (assuming your browser supports it):

```
/* style.css */
catalog {
   display: block;
   width: 500px;
   margin: 20px auto;
   border: 1px solid #ccc;
   padding: 20px;
}

book {
   display: block;
```

```
    margin-bottom: 15px;
    border-bottom: 1px dashed #ccc;
    padding-bottom: 10px;
}

title {
    font-size: 1.2em;
    font-weight: bold;
}

author {
    font-style: italic;
}
```

How to View in Browser

Save the files:

Save the DTD as books.dtd.

Save the XML as books.xml.

Save the CSS as style.css.

Make sure all three files are in the same directory.

Link the CSS in your XML:

Add the following line within the <catalog> element in your XML file:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

Open in Browser:

Open the books.xml file in a modern web browser. If your browser supports direct XML styling, you should see the catalog rendered with the CSS styles applied.

Important Notes

Browser Support: Direct XML styling with CSS is not universally supported. For broader compatibility, you'd typically use XSLT to transform your XML into HTML and then style the HTML with CSS.

XSLT: If you need to use XSLT, you'll create an XSLT stylesheet (transform.xsl) to define the transformation rules. Then, you'd link it in your XML using <?xml-stylesheet type="text/xsl" href="transform.xsl"?>.

Experiments 12:

12. Design an XML document and display it in browser using XSL.

1. XML Document (employees.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="employees.xsl"?>
<employees>
  <employee>
    <name>John Doe</name>
    <department>Sales</department>
    <salary>50000</salary>
  </employee>
  <employee>
    <name>Jane Smith</name>
    <department>Marketing</department>
    <salary>60000</salary>
  </employee>
  <employee>
    <name>David Johnson</name>
    <department>Engineering</department>
    <salary>75000</salary>
  </employee>
</employees>
```

XSL Stylesheet (employees.xsl)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <head>
```

```
      <title>Employee List</title>
    </head>
    <body>
      <h2>Employee Information</h2>
      <table>
        <tr>
          <th>Name</th>
          <th>Department</th>
          <th>Salary</th>
        </tr>
        <xsl:for-each select="employees/employee">
          <tr>
            <td><xsl:value-of select="name"/></td>
            <td><xsl:value-of select="department"/></td>
            <td><xsl:value-of select="salary"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Experiments 13:

13. Design XML Schema and corresponding XML document.

1. XML Schema (library.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
    <xs:element name="library">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="book" maxOccurs="un="title" type="xs:="isbn"
type="xs:string"/>="publicationYear" type="xs:gYear"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

2. XML Document (library.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="library.xsd">

    <book>
        <title>The Hitchhiker's Guide to the Galaxy</title>
        <author>Douglas Adams</author>
        <isbn>978-0-345-39180-3</isbn>
        <publicationYear>1979</publicationYear>
    </book>

    <book>
        <title>1984</title>
        <author>George Orwell</author>
        <isbn>978-0-451-52493-5</isbn>
        <publicationYear>1949</publicationYear>
    </book>

</library>
```

Explanation

XML Schema (library.xsd)

Defines the structure and data types allowed in the XML document

library is the root element

book elements can appear multiple times (maxOccurs="unbounded") within library

Each book has title, author, isbn, and publicationYear

Data types are specified (e.g., xs:string, xs:gYear)

XML Document (library.xml)

Adheres to the structure defined in the schema

xsi:noNamespaceSchemaLocation attribute links the XML document to the schema for validation

Key Points

Validation: The XML Schema ensures that the XML document conforms to the defined structure and data types.

Data Integrity: Using a schema helps maintain data consistency and prevents errors in the XML data.

Tooling: Many XML editors and processors can leverage schemas for validation, auto-completion, and other helpful features

Experiments 14:

14 Write a Basic PHP program to display "Hello World" using PHP programming.

```
<?php
echo "Hello World";
?>
```

Explanation:

<?php: This opening tag indicates the start of PHP code.

echo "Hello World";: The echo statement is used to output text or variables to the browser. In this case, it outputs the string "Hello World".

?>: This closing tag indicates the end of PHP code.

How to Run this PHP Program:

Save the code: Save the code in a file with a .php extension (e.g., hello.php).

Place it on a web server: You need a web server with PHP installed to run PHP scripts. If you're learning, you can set up a local development environment like XAMPP or WAMP.

Access the file through a web browser: Open your web browser and navigate to the URL where you placed the hello.php file (e.g., http://localhost/hello.php).

The browser will execute the PHP code on the server, and the output ("Hello World") will be sent back to the browser and displayed on the page.

Important Notes

PHP tags: The <?php and ?> tags are essential to tell the server that the code between them should be interpreted as PHP.

Semicolon: Each PHP statement ends with a semicolon (;).

Case sensitivity: PHP is case-sensitive for variable names, but not for keywords like echo.