"In the name of Allah, the Most Gracious, the Most Merciful"

International Islamic University Chittagong (IIUC)

Department of Computer and Communication Engineering (CCE)

'Lab Manual'

Course Code: CCE-1206

Course Title: Web Programming Sessional

Prepared by:

Mohammad Nadib Hasan (MNH)

Lecturer, Department of CCE, IIUC.

Table of Contents

LAB# 8, 9 and 10		**Java Script**	**39-48**
		Introduction to JavaScript.	
		JavaScript Operators, Loops, Function and object.	
L-10	**S-1**	**DOM Interaction and Manipulation in JS [Part-1]**	
	S-2	**DOM Interaction and Manipulation in JS [Part-2]**	
LAB# 11 and 12		**PHP, Web Server and MySQL**	**49-61**
L-11	**S-1**	**PHP and Web Server: Installation and Configuration.**	
	S-2	**Introduction to PHP: Syntax, Variables, Function, Loops and Control Structures.**	
L-12	**S-1**	**Introduction to Databases with PHP: DB Connection, Data Query, PHP Session, Cookies & Redirection.**	
	S-2	**PHP: Form (Login, Registration, Feedback) to collect data and store it in database.**	

COURSE DESCRIPTION:

ISCED Code: 0613	**Course Code: CCE-1206**	**Course Title: Web Programming Sessional**	
Credit Hours: 1.5	**Contact Hours: 3 CH per Week**	**Prerequisite Course: CCE-1106**	
Course Assessments	**CIE: Continuous Internal Evaluation**		**40 – 60 Marks**
	SEE: Semester End Examination		**60 – 40 Marks**

Contents	CLOs	Lecture	Practical
HTML: Basics, Elements, Attributes, Headings, Paragraphs, Formatting, Links, Head, Images, Tables, Lists, Blocks, Layout, Forms, multimedia tags.	CLO – 1 & 2	4	8
CSS: Introduction, Syntax, Selectors - Id & Class, Styling Backgrounds, Text, Fonts, Links, Box Model- Border, Outline, Margin, Padding, Grouping/Nesting, Dimension, Display, Positioning, Floating, Align, Navigation Bar, Image Gallery.		3	6
JavaScript: Introduction, Output, Statements, Comments, Variables, Data Types, Objects, Functions, Operators, Comparisons, Conditions, Loop, Errors, DOM Introduction and manipulation, Number, String, Date, Array, Window, Screen, Navigator, Popup Alert.		4	8
PHP: Syntax, Variables, String, Operators, If…Else, Switch, Arrays, While Loops, For Loops, Functions, Date, Include, File Upload, Cookies, Forms, $_GET and $_POST methods.		4	8
Total		15	30

INTRODUCTION TO THE COURSE:

Internet and Web become an integral part of human life. It exists in every possible dimension which makes this art essential to learn. This course studies both theoretical and practical approach to Web Engineering. It provides a highly-interactive introduction to Web Programming using client-side technologies (HTML, CSS and JavaScript) and server-side technologies (PHP) to create web pages and web applications. A list of problems is also provided at the end of each lab session. Please go through this lab manual sequentially and follow the ***general guidelines*** carefully.

OBJECTIVE(S) OF THIS COURSE:

- ✓ To understand the concept of Web Application Development and its Architecture.
- ✓ To understand the Essentials of Web Application Development.
- ✓ To understand and practice web page designing techniques.
- ✓ To understand and practice embedded dynamic scripting on client-side Internet Programming.
- ✓ To understand the differences between client side & server-side technologies to develop Web Application.

Course Learning Outcomes (CLOs):

CLOs	Course Learning Outcomes (CLOs): Upon the successful completion of the course, students will be able to	Bloom's Taxonomy Domain/Level	Program Learning Outcomes (PLOs)
CLO-1	Understand functionality of Web programming and its applications.	Cognitive (Understand)	PLO – 1
CLO-2	Design and develop interactive, client-side, server-side executable web applications.	Cognitive (Evaluation) Psychomotor	PLO – 4

ASSESSMENT METHODS:

Teaching Learning Strategy:

Face-to-Face Learning

- Lecture
- Experiment

Self-directed Learning

- Preparation for Lab Reports
- Preparation for Lab Test & Quiz
- Engagement in Project / Assignment

Student Assessment Methods:

Formative Assessment

- Continuous Assessment (Experiment Conduction, Lab Report and Lab Viva)

Summative Assessment

- Final Quiz / Final Lab Examination
- Lab Test / Lab Performance / Project Show

CLOs with Weighting of Assessments:

CLOs	Assessment Method		(%)
	Attendance		10%
CLO-1 and CLO-2	Continuous Assessment	Experiment Conduction	10%
		Lab Report	20%
		Lab Viva	10%
	Final Lab Quiz / Final Lab Examination		30%
	Final Lab Performance / Test & Viva		20%

Rubrics followed to Evaluate the Lab Courses:

Attendance:

Attendance	Awarding marks
90% and above	10
85% to less than 90%	9
80% to less than 85%	8
75% to less than 80%	7
70% to less than 75%	6
65% to less than 70%	5
60% to less than 65%	4
less than 60%	0

Continuous Assessment (Experiment Conduction, Lab Viva and Lab Report):

Parameter	Allocated Marks	Low	Medium	High
Experiment Conduction	**10**	The student has not performed anything during laboratory periods	The student has given satisfactory performance during laboratory periods	The student has given excellent performance or has completed all the tasks given during laboratory periods.
		0 Mark	**1 – 5 Marks**	**6 – 10 Marks**
Lab Viva	**10**	**Low**	**Medium**	**High**
		The student was not able to answer anything during viva-voce.	The student was able to answer a few questions during viva-voce	The student was able to answer all the questions during viva voce.
		0 Mark	**1 – 5 Marks**	**6 – 10 Marks**
Lab Performance	**10**	The student was not able to perform the job, given during the semester laboratory examination.	The student was partially able to perform the job during the semester laboratory examination.	The student was able to perform the job accurately during the semester laboratory examination.
		0 Mark	**1 – 5 Marks**	**6 – 10 Marks**

Lab Report Rubric:

	Parameter	Fail (0)	Poor (2)	Fair (4)	Good (7)	Excellent (10)
Lab Report	**Report format and quality (5)**	No submission / No effort exhibited and No attention to detail evident	Directions were not followed and report contains many errors.	Report is somewhat organized with some spelling or grammatical errors.	Report is well organized and cohesive but contains minor errors in format or procedures.	Lab report submitted as directed, and on time. Directions were followed. Report is well organized and cohesive and contains no mechanical errors. Presentation seems polished.
	Experiment Background, Results & Data Analysis (10)	No submission / No effort exhibited.	Introduction and background are insufficient or missing entirely. Data is missing, inaccurate, or not analysed effectively.	Presents an introduction and background information, but it lacks clarity and relevance to the experiment. Presents data with several inaccuracies or significant organizational problems. Data analysis is limited or contains major errors.	Provides a sufficient introduction and background information but lacks some depth or clarity. Presents mostly accurate data with some organizational issues. Analyses data adequately but with minor errors or inconsistencies.	Clearly articulates the purpose, significance, and relevant background information of the experiment. Presents accurate, well-organized, and comprehensive data. Analyses data effectively using appropriate methods.
	Discussion & Conclusion (5)	No submission / No effort exhibited	Does not effectively connect results to	Attempts to connect results to objectives but with	Connects the results to the objectives but lacks	Demonstrates a clear understanding of the results and connects

			objectives, and conclusions are absent or inaccurate.	significant gaps or inaccuracies. Conclusions are vague or unsupported.	depth or thorough analysis. Conclusions are drawn but lack insight or depth.	them to the experiment's objectives. Offers insightful interpretations and conclusions supported by evidence.

GRADING POLICY:

Numerical grade Marks%	Letter Grade (LG)	Grade Point (GP/unit)	Remarks/ Status
80-100	A+ (A plus)	4.00	Excellent
75 to less than 80	A (A regular)	3.75	Very good
70 to less than 75	A- (A minus)	3.50	
65 to less than 70	B+ (B plus)	3.25	Good
60 to less than 65	B (B regular)	3.00	
55 to less than 60	B- (B minus)	2.75	Satisfactory
50 to less than 55	C+ (C plus)	2.50	
45 to less than 50	C (C regular)	2.25	Not Satisfactory
40 to less than 45	D (D regular)	2.00	
less than 40	F	0.00	Fail

GENERAL GUIDELINES:

Dear Students,

Welcome to Web Programming Lab.

- For the practical works of Web Programming Course, you have to complete CCE-1206 lab (3 CH each) activities throughout the course. This lab manual will guide you to prepare for making and submission of lab reports. Further, it helps you to understand practically about the knowledge of Web Programming. You can use this lab manual as the base reference during your lab.
- You have to submit lab report of previous lab into corresponding next lab during when your instructor shall take necessary lab performance for each lab works. For your reference, *"**how to write a complete CCE-1206 lab report?"*** is being prepared as sample lab report in this manual. For the rest of your labs, please follow the reporting style as provided.

- Your lab report to be submitted should include at least the following topics.
    1. **Cover Page**
    2. **Experiment No**
    3. **Experiment Name**
    4. **Objectives**
    5. **Problem Statement/Theory**
    6. **Coding (Source Code)**
    7. **Output (compilation, debugging & testing)**
    8. **Discussion & Conclusion.**

- You should attempt all ***Experiments*** given in the list ***lab wise***.
- You may seek assistance in doing the lab Experiments from the concerned lab instructor. Since the assignments have credits, the lab instructor is obviously not expected to tell you how to solve these, but you may ask questions concerning the Web Programming or a technical problem.
- For each program you should add comments above each function in the code, including the main function.
- The comment block above the main code should describe the purpose of the program. Proper comments are to be provide where and when necessary, in the coding.
- The code should be interactive, general and properly documented with realInput/ Output data.
- If two or more submissions from different students appear to be of the same origin (i.e., are variants of essentially the same program), none of them will be counted.
- You are strongly advised not to copy somebody else's work.
- It is your responsibility to create a separate directory to store all the programs,so that nobody else can read or copy.
- As soon as you have finished a lab Experiment, contact your lab instructor in order to get the Experiment evaluated and also get the signature from him/her on the lab performance book/sheet.

1. HOW TO WRITE A COMPLETE LAB REPORT?

Page VIII to IX shows a sample of a complete lab report.

الجامعة الإسلامية العالمية شيتاغونغ
International Islamic University Chittagong
Department of Computer and Communication Engineering (CCE)

LAB REPORT

Course Title : Web Programming Sessional
Course Code : CCE-1206

Session Topic:

Submitted By

Name :
ID No :
Semester :
Section :

Date of Experiment:
Date of Submission:

Remarks

Submitted To

Engr. Mohammad Nadib Hasan
Lecturer, Dept. of CCE, IIUC.

Your lab report to be submitted should include at least the following topics.

1. **Cover Page**
2. **Experiment No**
3. **Experiment Name**
4. **Objectives**
5. **Problem Statement/Theory**
6. **Coding (Source Code)**
7. **Output (compilation, debugging & testing) and**
8. **Discussion & Conclusion.**

PART II: LAB SESSIONS

Lab #1: Introduction to Web Programming

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 01

Experiment Name: Installation of Visual Studio Code.

Theory: Visual Studio Code, commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework for Windows, Linux, and macOS. It features support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

Procedure: Follow the steps below to install VS Code on Windows:

Step 1: Visit https://code.visualstudio.com and click on the "Windows Download" option. By clicking the download option, an exe file (approximately 90 MB) will be downloaded, and the process should not take much time.

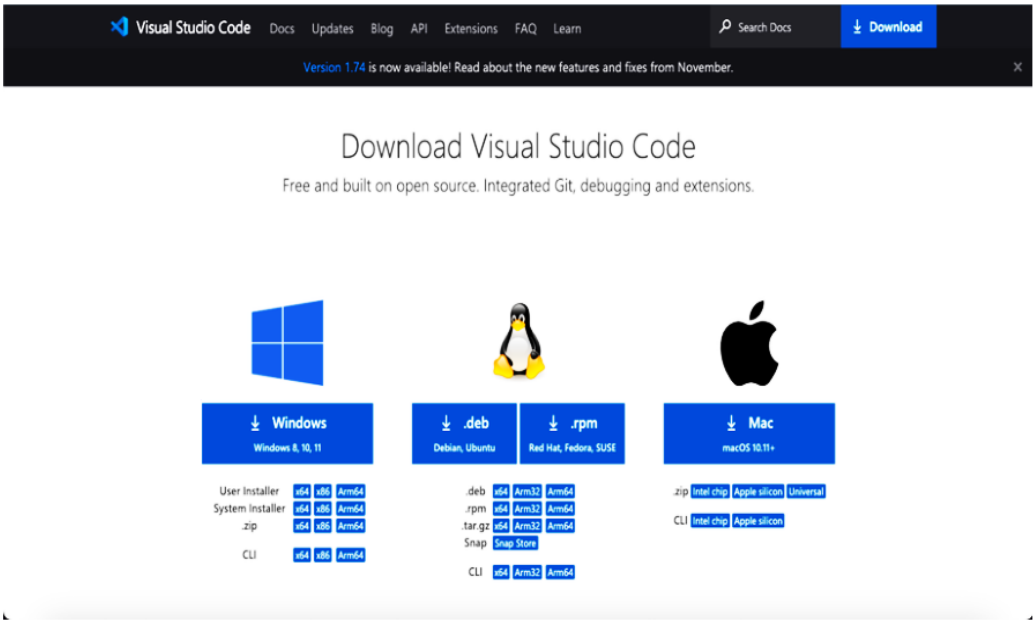


Figure - 1

Step 2: After the file is downloaded, open it, and the following screen will appear. Check the "I accept the agreement" option and proceed by clicking the "Next" button.

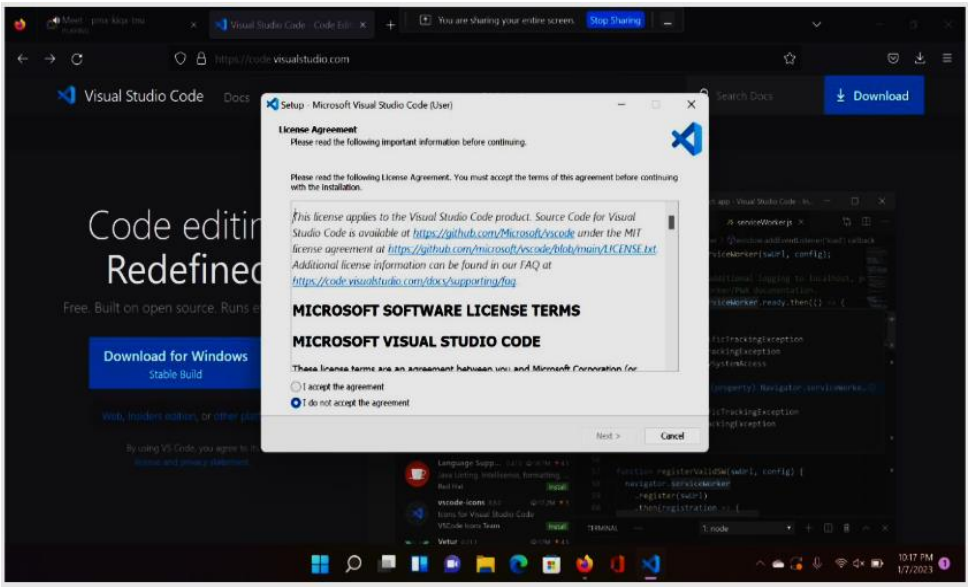


Figure - 2

Step 3: The next screen is for selecting the file location. You can avoid changing the default location by simply clicking the "Next" button.

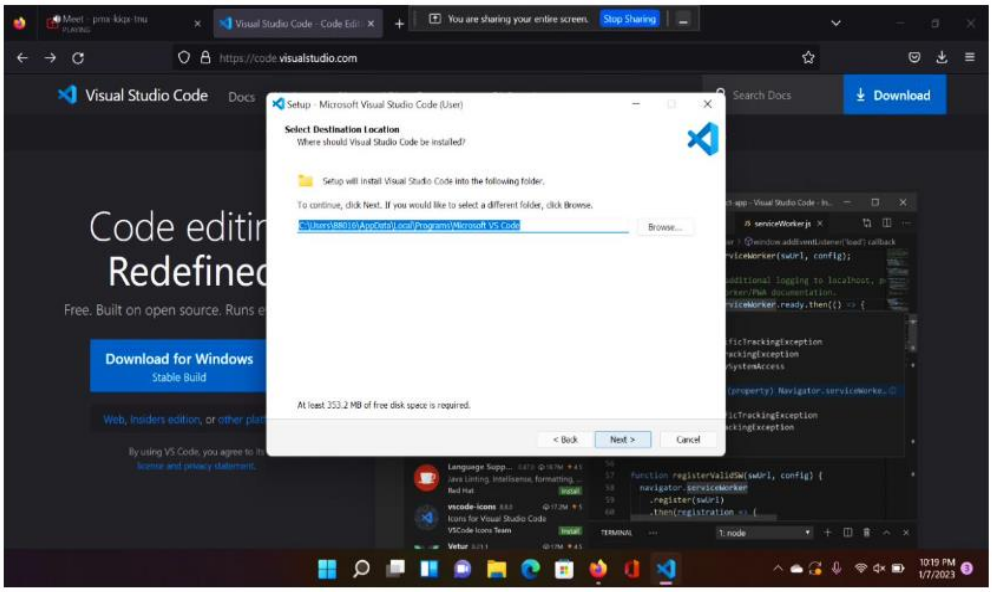


Figure - 3

Step 4: The subsequent screen is for creating a shortcut in the start menu bar. You can click the "Next" button without making any changes.

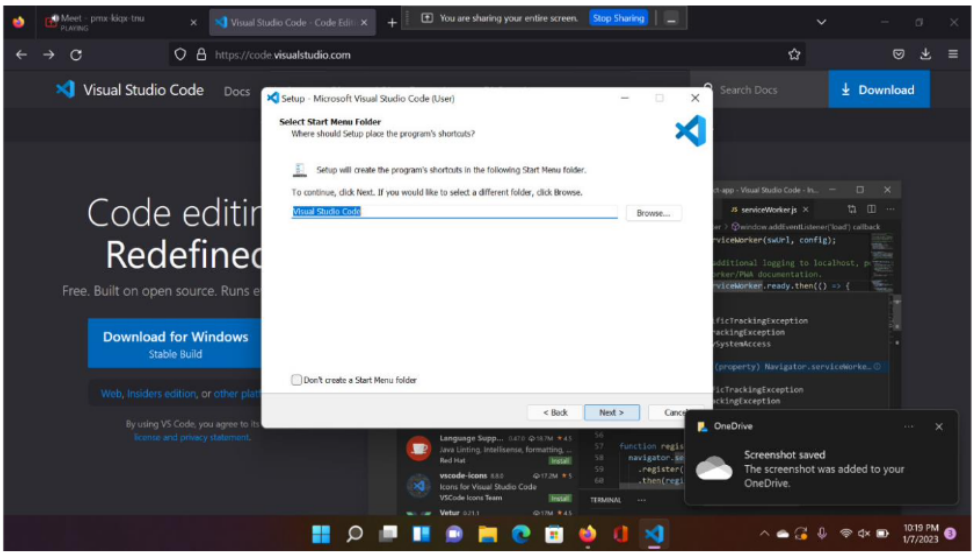


Figure - 4

Step 5: This screen will display the available additional tasks for VS Code. If you want to avoid changing anything, click the "Next" button.

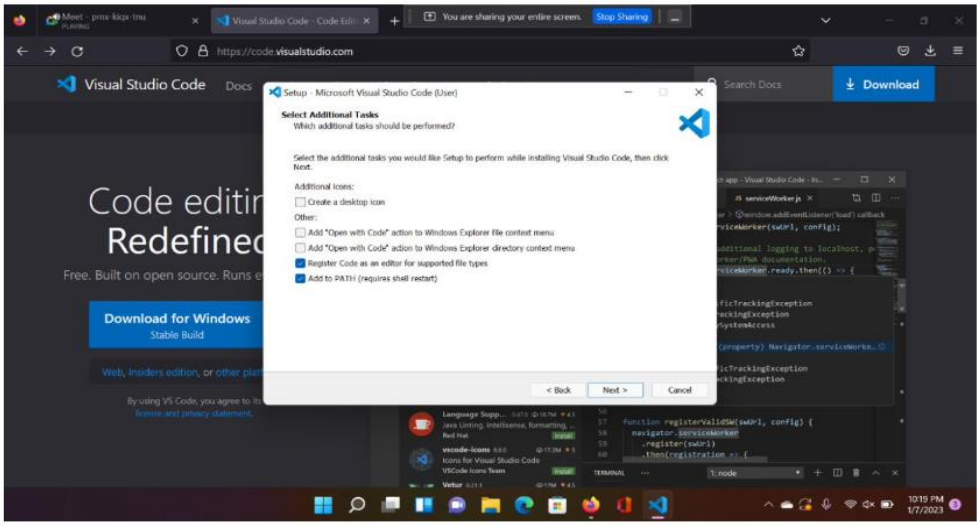


Figure - 5

Step 6: The next screen will show a description and an "Install" button. Click the "Install" button, and the installation will be complete within a minute.

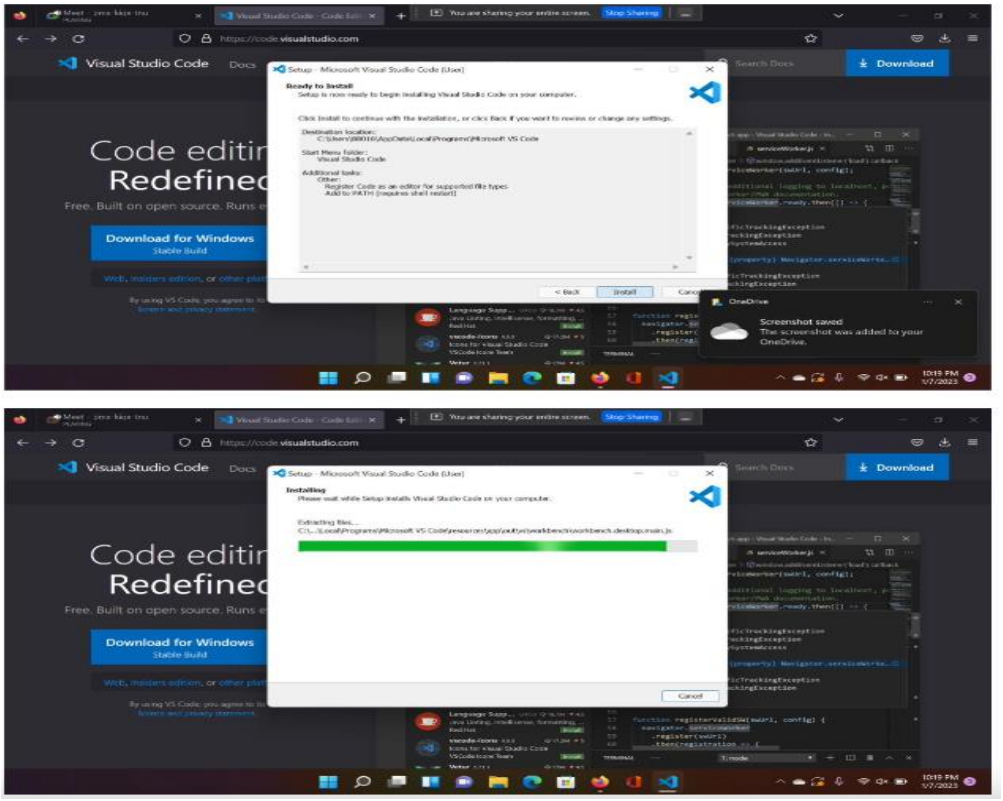


Figure - 6

Step 7: After the installation is finished, this screen will appear. Click the "Finish" button to complete the installation process, and you are ready to open Visual Studio Code.

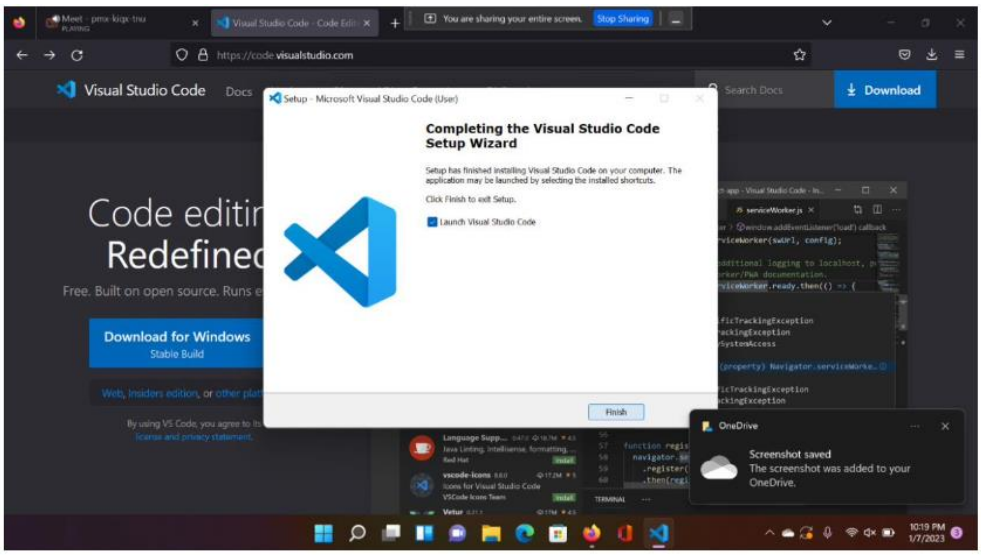


Figure - 7

Step 8: Run the software and explore its interface.

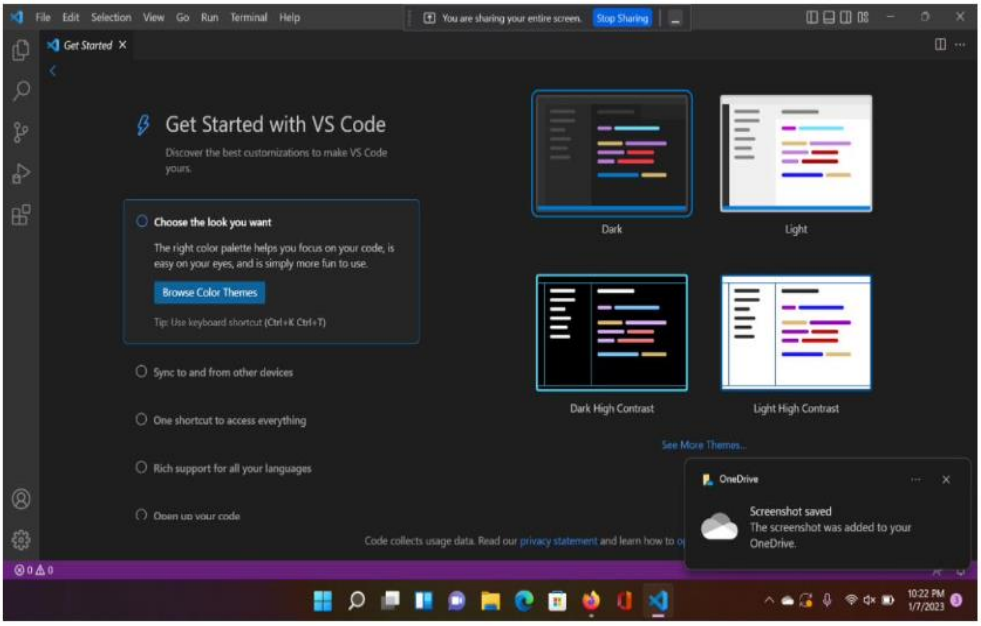


Figure - 8

Conclusion: In conclusion, the installation process for Visual Studio Code on Windows is straightforward and user-friendly. By following a series of simple steps, you can quickly download, install, and set up the software.

Lab #2: The Building Blocks of HTML and Essential Tags

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 02

Experiment Name: Introduction to HTML and its Basic elements.

Theory: HTML (Hypertext Markup Language) is the standard markup language used to create the structure and content of web pages. It consists of a series of elements that define the structure of the content, such as headings, paragraphs, lists, links, images, and more.

To write HTML code:

1. Open a text editor (e.g., Visual Studio Code, Notepad).
2. Create a new HTML file.
3. Save the file with a ".html" extension (e.g., index.html, contact.html, biography.html).
4. Open the HTML file in a web browser to view the result.

Understanding HTML elements:
Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First HTML Page</title>
  </head>
  <body>
    <h1>Hello, World! </h1>
    <p>This is my first HTML page. </p>
  </body>
</html>
```

Break down the HTML structure into elements:

- <!DOCTYPE html> → Document type declaration
- <html> → Root element
- <head> → Contains meta-information about the HTML document
- <title> → Sets the title of the document (shown in the browser tab)
- <body> → Contains the content of the HTML document
- <h1> → Header 1 (a heading tag)

- <p> → Paragraph

Text and Headings: Experiment with different heading tags (<h1>, <h2>, <h3>) and paragraphs (<p>).

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Text and Headings</title>
  </head>
  <body>
    <h1>Main Heading</h1>
    <h2>Subheading 1</h2>
    <p>This is a paragraph of text.</p>
    <h3>Subheading 2</h3>
    <p>Another paragraph here.</p>
  </body>
</html>
```

Hyperlink: Introduce the <a> (anchor) tag to create hyperlinks.

Example:

```
<p>
   Visit my <a href="https://www.example.com">website</a> for more information.
</p>
```

Block-Level and Inline Elements:

- Block-level elements start on a new line and take up the full width (e.g., ,<ul>, <li>,<hr>, <p>, <h1>).
- Inline elements do not start on a new line and only take up as much width as necessary (e.g., , <a>, <img>,
,).

Text Formatting Tags: Explore formatting tags for text:

- → Bold text
- → emphasize text
- <u> → Underlined text

Example:

```
<p>This is <strong>strong</strong> text.</p>
<p>This is <em>emphasized</em> text.</p>
<p>This is <u>underlined</u> text.</p>
```

Semantic Tags: Introduce semantic tags for better structure and meaning: <header>, <nav>, <section>, <article>, <aside>, <footer>

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Semantic Tags</title>
  </head>
  <body>
    <header>
      <h1>My Personal Website</h1>
    </header>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
      </ul>
    </nav>
    <section>
      <article>
        <h2>Article Title</h2>
        <p>Article content goes here.</p>
      </article>
      <aside>
        <h3>Related Links</h3>
        <ul>
          <li><a href="#">Blogs</a></li>
          <li><a href="#">Contact</a></li>
        </ul>
      </aside>
    </section>
    <footer>
      <p>&copy; 2024 My Personal Website</p>
    </footer>
  </body>
</html>
```

Section B: Lab Experiments - Demonstrated by students

Lab Task:

- Create a webpage. Use block-level elements such as headings, paragraphs, and lists to structure your content.
- Apply text formatting tags (e.g., <i>, <u>, <b>,) to emphasize certain parts of your content.
- Add a navigation bar using <nav> and create hyperlinks <a> for each section of your webpage within the navigation bar.
- Use proper semantic tags in each page (<header>, <section>, <article>, , <footer>).

Lab #3: HTML: Table and List

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 03

Experiment Name: Write HTML Program for Tables.

Theory: A Table is an arrangement of rows and columns. Anyone can create a table by knowing the basics of HTML. A table is defined by using the <table> tag in HTML.

Steps to create an HTML Table:

- Start creating tables using the <table> tag and use </table> at the end.
- Then provide a name for this table with the <caption></caption> tag.
- Inside the starting and ending points define the table head and table body using <thead></thead> and <tbody></tbody> tag.
- Inside <thead> and <tbody> tag define the table row using <tr> tags.
- Inside <thead>: define columns inside each row (<tr>) using <th> tags.
- Inside <tbody>: define columns inside each row (<tr>) using <td> tags.

Source Code:

```
<table border="2" cellspacing="3" cellpadding="3">
    <thead>
        <tr>
            <th colspan="2">Employee</th>
            <th colspan="2">Salary</th>
        </tr>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Basic Salary</th>
            <th>House Rent</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>EG789</td>
            <td>John Doe</td>
            <td>30000</td>
            <td>5000</td>
        </tr>
        <tr>
            <td>EG421</td>
            <td>Jane Smith</td>
            <td>35000</td>
            <td>6000</td>
        </tr>
        <tr>
            <td colspan="2">Total Salary</td>
            <td>75000</td>
            <td>11000</td>
        </tr>
    </tbody>
</table>
```

[**Note**: *"cellspacing" defines space outside of each cell. "cellpadding" defines space inside of each cell. "border" defines the border in each cell]. "rowspan" allows a cell to span multiple rows and "colspan" allows a cell to span multiple columns.*

Output:

Employee		Salary	
ID	**Name**	**Basic Salary**	**House Rent**
EG789	John Doe	30000	5000
EG421	Jane Smith	35000	6000
Total Salary		75000	11000

Figure: HTML Table

Section B: Lab Experiments - Demonstrated by students

Lab Task - 1: Create your class routine using HTML Table tags.

Lab Task - 2: Create a table similar to the table given below:

Heading	Students	Details		
	Id	Name	Department	Roll Number
Student List	1	Victor	Computer Science	12345
	2	Williams	Electronics	23456
	3	Harry	Electrical	34567
	4	Rick	Civil	45678

Experiment No: 04

Experiment Name: Introduction to HTML Lists.

Theory: HTML Lists are used to specify lists of information. All lists may contain one or more list elements. There are three basic types of HTML lists:

1. Ordered List (ol)
2. Unordered List or Bulleted List (ul)
3. Description List or Definition List (dl).

1. HTML Ordered List (ol):

- In HTML, all the list items in an ordered list are marked with numbers by default instead of bullets.
- An HTML-ordered list starts with the <ol> tag and ends with the tag.
- And, inside the <ol> tags, we have to define the lists using tags.
- Inside the tags we have to define the list contents.

Default Ordered List:

Source Code:

```
<body>
<h2>List of Fruits</h2>
   <ol>
      <li>Apple</li>
      <li>Mango</li>
      <li>Banana</li>
      <li>Grapes</li>
      <li>Orange</li>
   </ol>
</body>
```

Output:

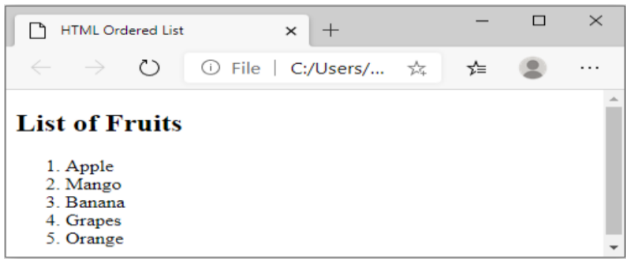


Figure: Ordered List

Custom Ordered List:

Source Code:

```
<h2>List of Fruits</h2>

<ol type="i">
    <li>Apple</li>
    <li>Mango</li>
    <li>Banana</li>
</ol>
```

Output:

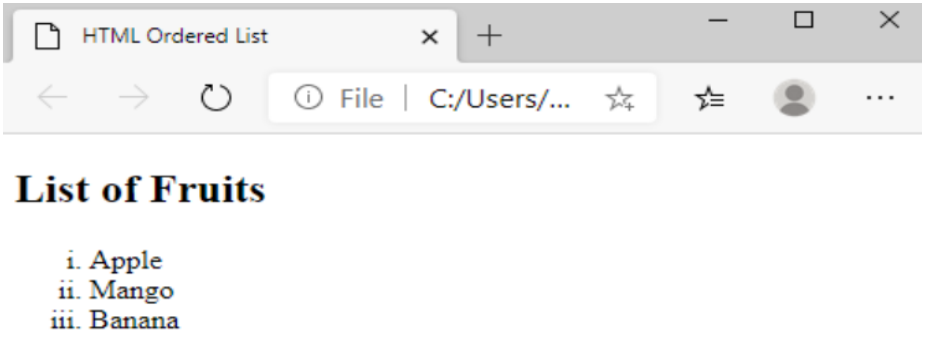


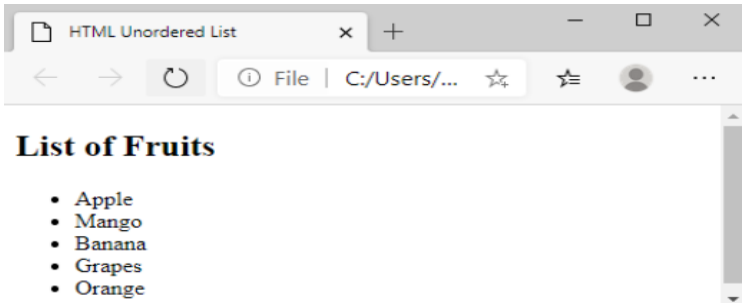
Figure: Ordered List

Unordered List:

Source Code:

```
<h2>List of Fruits</h2>
   <ul>
      <li>Apple</li>
      <li>Mango</li>
      <li>Banana</li>
      <li>Grapes</li>
      <li>Orange</li>
   </ul>
```

Output:



HTML Description List / Definition List:

- <dl> (Definition list) tag – Start tag of the definition list
- <dt> (Definition Term) tag – It specifies a term (name)
- <dd> tag (Definition Description) – Specifies the term definition
- </dl> tag (Definition list) – Closing tag of the definition list

Source Code:

```
<dl>
   <dt><b>Apple</b></dt>
   <dd>A red colored fruit</dd>
   <dt><b>Honda</b></dt>
   <dd>A brand of a car</dd>
   <dt><b>Spinach</b></dt>
   <dd>A green leafy vegetable</dd>
</dl>
```

Output:

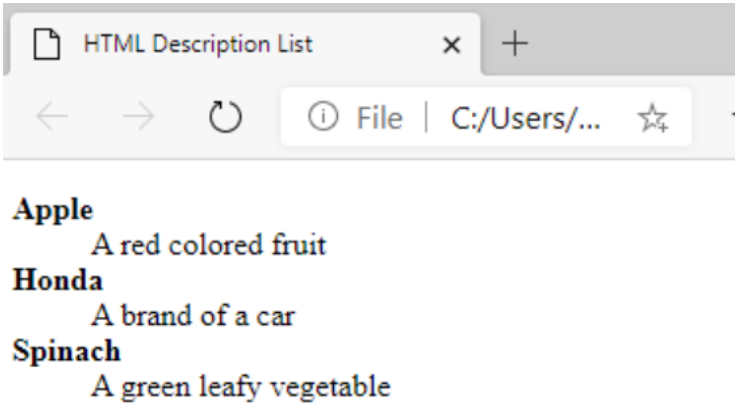


Figure: Description List

Section B: Lab Experiments - Demonstrated by students

Lab Task - 1: Create HTML lists similar to the following lists:

1. **Cart Order:**
   A. Book
   B. Pen
   C. T-Shirt
   D. Jeans
2. **Vehicles List:**
   a. Cycle
   b. Car
   c. Bus
   d. Taxi
3. **List of Programming Languages:**
   - C
   - C ++
   - Java
   - JavaScript
   - Rust
4. **List of Database:**
   - MySQL
   - MongoDB
   - PostgreSQL
   - GraphQL

[***hints:*** search list style types]

Lab Task - 2: Describe the information, and qualities about the listed person with a title using HTML definition list:

- ❖ Engr. Md Razu Ahmed
- ❖ Dr. Mohammad Saifuddin
- ❖ Engr. Jiabul Hoque
- ❖ Engr. Mohammad Nadib Hasan
- ❖ Engr. Md Humayan Kabir
- ❖ Engr. Hassan Zaki
- ❖ Engr. Areez Hafiz Md Zahid
- ❖ Engr. Zarin Tanzim

[***hints:*** You can get information about them from this link: *https://www.iiuc.ac.bd/cce/faculty*]

Lab #4: HTML: Form, Multimedia, Frames & iFrames

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 05

Experiment Name: Create a form using HTML.

Theory:

Form: An HTML form is a section of a web page that contains various elements such as input fields, buttons, checkboxes, radio buttons, and more, allowing users to submit data to a web server for processing.

Input: <input> tags are used to create various types of form controls that allow users to input data. The "type" attribute of the <input> tag determines the type of input control to be displayed, such as text input, checkboxes, radio buttons, and more.

Basic Input types:

- Text Input:

```
<input type="text" />
```

- Email Input:

```
<input type="email" />
```

- Password Input:

```
<input type="password" />
```

- Text Area Input:

```
<textarea name="details" />
```

- Radio Input:

```
<input type="radio" name="gender" value="female" />
<input type="radio" name="gender" value"male" />
```

- Checkbox Input:

```
<input type="checkbox" name="terms-and-condition"/>
```

- Select and Options Input:

```
<select>
  <option value="autumn-2023">Autumn 2023</option>
  <option value="spring-2024"> Spring 2024</option/>
  <option value="autumn-2024">Autumn 2024</option>
</select>
```

- Form submit button:

```
<input type="submit" value="Login" />
```

Source Code:

```
<form action="/collect-tour-info" align="center">
  <h1>Tour Registration</h1>
  <label>First Name</label>
  <input type="text" placeholder="First Name"> <br><br>
  <label>Last Name</label>
  <input type="text" placeholder="Last Name"><br><br>
  <label>Email</label>
  <input type="cmail" placeholder="your@email.com"> <br><br>
  <label>Password</label>
  <input type="password" placeholder="Your Password"><br><br>
  <label>Gender</label>
  <input type="radio" value="Male"/> Male
  <input type="radio" value="Female" /> Female<br><br>
  <label>Country</label>
  <select>
    <option>Bangladesh</option>
    <option>United States</option>
    <option>India</option>
  </select>
  <br><br>
  <lable>Image</lable>
  <input type="file"><br><br>
  <input type="Submit">
  <input type="Reset">
</form>
```

Output:

Figure: Basic HTML form

Section B: Lab Experiments - Demonstrated by students

Lab Task - 1: Make an HTML form to collect the information for admitting patients in a hospital.

Lab Task - 2: Suppose you are a university teacher; a student wants to get admission to your university. Create a form by using HTML to collect necessary admission information from him.

Experiment No: 06

Experiment Name: Introduction to HTML multimedia tags:

A. Create a section to display images using <img> tag
B. Create a section to play video files using <video> tag
C. Create a section to play audio files using <audio> tag
D. Create a section to display another webpage content / YouTube content using <iframe> tag

Theory: HTML multimedia tags are used to embed and manage various types of media content like images, audio, and video on web pages. These tags include `*<img>*` for images, `*<audio>*` for sound, and `*<video>*` for video playback, providing seamless integration of multimedia elements. With attributes such as controls, autoplay, and source options, these tags allow developers to customize the user experience. They play a key role in creating interactive, dynamic, and engaging websites.

1. Embedding image using <img> tag:

Source Code:

```
<img
 src="https://i.ibb.co/Jpmpd9P/chris-stenger-fv-Jwch-RL6xw-unsplash-1.jpg"
 width="750"
 height="450"
 alt="Aesthetic view of a Lake with mountains"/>
```

- ❖ width and height: Set the width and height of the image.
- ❖ <source>: Specifies the image file.
- ❖ alt="..." refers to the description of the image (optional)

Output:

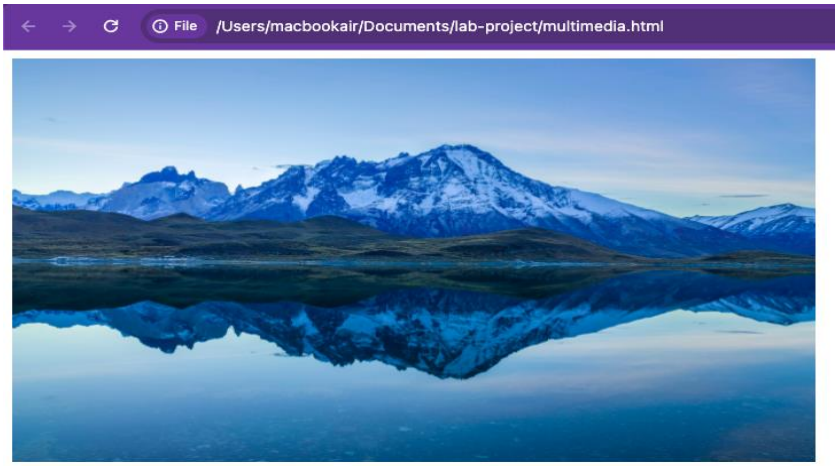


Figure: Embedded image

Embedding video using <video> tag:

The <video> tag is used to embed video content on a web page.

Source Code:

```
<video width="640" height="360" controls>
   <source
        src="videos/presentation.mp4"
        type="video/mp4"/>
</video>
```

- ❖ width and height: Set the width and height of the video player.
- ❖ controls: Adds playback controls (play, pause, volume, etc.).
- ❖ <source>: Specifies the video file and its type.

Output:

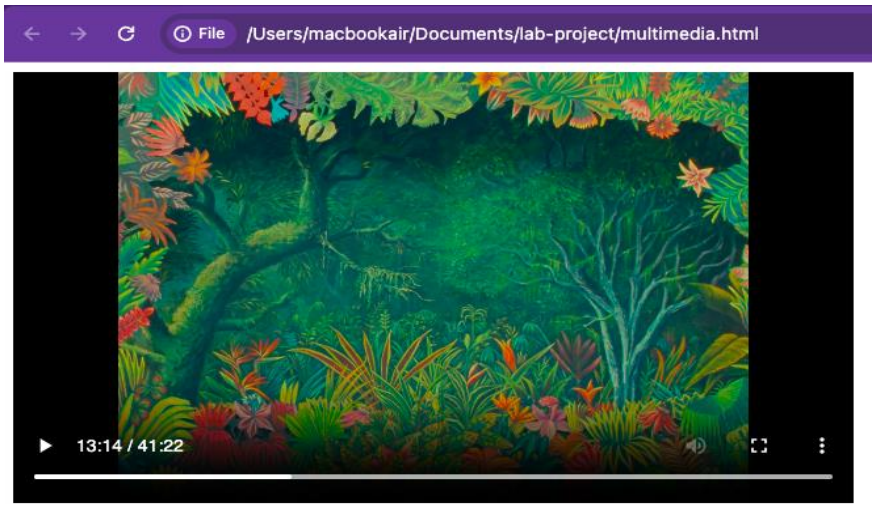


Figure: Embedded video

Embedding audio using <audio> tag:

The <audio> tag is used to embed audio content on a web page.

Source Code:

```
<audio controls>
  <source src="sound/playback.mp3" type="audio/mpeg"/>
</audio>
```

- ❖ controls: Adds playback controls for audio (play, pause, volume, etc.).
- ❖ <source>: Specifies the audio file and its type.

Output:

Figure: Embedded audio

Embedding external document using <iframe> tag:

The <iframe> (inline frame) tag is used to embed an external document or webpage within the current HTML document.

Source Code:

```
<iframe width="560" height="315"
        src="https://www.youtube.com/embed/LLuolLkKT5E?si=uVyjvJSTNafATRJE"
        title="YouTube video player"
        frameborder="0"
        allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope;">
</iframe>
```

Output:

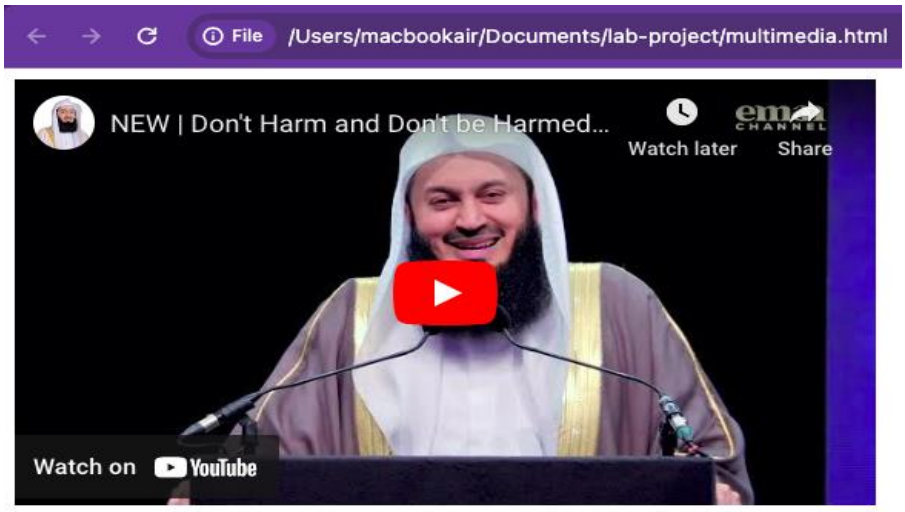


Figure: Embedded external document using iframe

Section B: Lab Experiments - Demonstrated by students

Lab Task: Create a simple website with 4 pages, each using a different HTML file: "image.html", "video.html", "audio.html", and "iframe.html". Add a header on each page with links so you can easily navigate between pages by clicking the links.

Lab #5: Introduction to CSS

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 07

Experiment Name: Introduction to CSS.

Theory: CSS (Cascading Style Sheets) is a language used to style and layout web pages. It controls the appearance of HTML elements, such as colors, fonts, spacing, and overall layout. By separating the content (HTML) from the design (CSS), developers can create visually appealing websites that are easier to maintain and update. CSS also allows for responsive designs, making web pages look good on different screen sizes and devices.

There are 3 ways to use CSS with HTML: Inline CSS, Internal CSS, External CSS

1. Inline CSS: Inline CSS is applied directly to an HTML element using the style attribute.

Example:

```
<p style="color: blue; font-size: 16px">This is a paragraph with inline CSS.</p>
```

2. Internal CSS: Internal CSS is defined within the <style> tag in the <head> section of the HTML document.

Example:

```
<html lang="en">
 <head>
  <style>
   h1 {
    color: green;
    font-size: 24px;
   }
  </style>
 </head>
 <body>
  <h1>This is a heading with internal CSS.</h1>
 </body>
</html>
```

3. External CSS: External CSS is defined in a separate CSS file and linked to the HTML document using the <link> tag.

Example:

(Inside index.html)

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <link rel="stylesheet" href="styles.css" />
 </head>
 <body>
  <h2>This is a heading with external CSS.</h2>
 </body>
</html>
```

Inside style.css:

```
h2 {
 color: red;
 font-size: 20px;
}
```

CSS Selectors: Selectors are patterns used to select and style HTML elements. There are various types of selectors, such as element selectors, class selectors, and ID selectors.

1. Element Selector: Selects all instances of a particular HTML element.

Example:

```
p {
 color: lightblue;
}
```

2. Class Selector: Selects elements with a specific class attribute.

Example:

```
.container {
 background-color: pink;
}
```

3. ID Selector: Selects a single element with a specific ID attribute.

Example:

```
#navbar {
 background-color: cyan; }
```

CSS Typography:

- color → sets font color
- Font-size → sets the size of the font
- font-weight → sets the thickness of the font
- Font-style → sets the font style

Example:

```
p {
 color: lightblue;
 font-size: 18px;
 font-weight: 700;
 font-style: italic;
}
```

CSS Box Model: The CSS box model describes the layout and spacing of elements. It describes how every element on a webpage is represented as a rectangular box with four distinct components: content, padding, border, and margin.

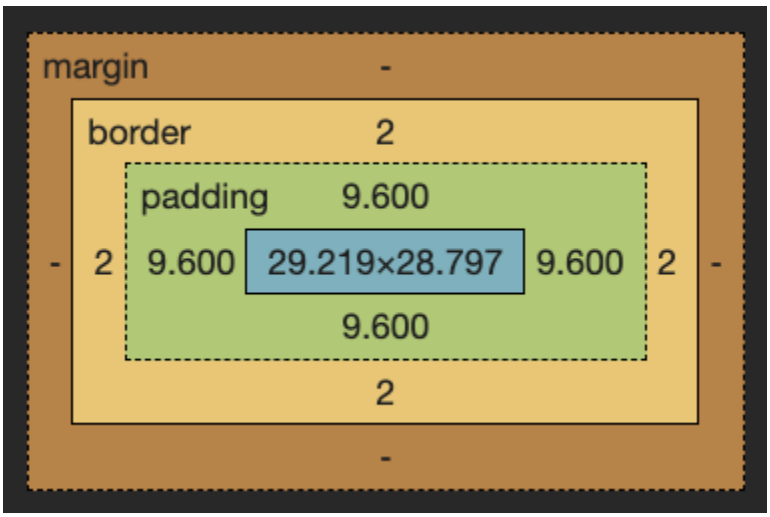


Figure: Box Model from dev-tool

1. Content: Content means the actual content. It includes the content's height and width.

2. Border: The border is the line that surrounds the content. It can have a specific width, style, and color.

3. Padding: Padding is the space between the content and the border. It helps control the internal spacing of the element.

4. Margin: The margin is the outermost layer. Margin provides space between the border of the content/element and surrounding elements. It helps control the external spacing of the element.

Example:

```
.box {
  width: 200px;
  height: 150px;
  padding: 20px;
  border: 2px solid gray;
  margin: 10px;
}
```

Section B: Lab Experiments - Demonstrated by students

Lab Task: Create a simple HTML webpage with at least three parts. Each part should contain Heading, Paragraph, List of Items, etc.

Apply the following styling using CSS:

- ➔ Use the class selector and ID selector to style each part differently.
- ➔ Adjust the box model properties (margin, padding, etc.) for selected elements.
- ➔ Set different font-size for headings and paragraphs.
- ➔ Use font-weight and font-style to modify the appearance of specific text.
- ➔ Apply a background color to one or more elements.

Lab #6: CSS Positioning

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 08

Experiment Name: Introduction to CSS Positioning.

Theory: CSS positioning is used to control the layout of elements on a web page. It allows developers to place elements precisely where they want using properties like '*static*', '*relative*', '*absolute*', '*fixed*', and '*sticky*'. Each type of positioning behaves differently, offering flexibility for organizing content. For example, '*relative*' moves an element to its normal position, while '*absolute*' places it based on its closest positioned ancestor. This helps create dynamic and responsive layouts.

1. Static Positioning: It is the default position behavior. If the position property isn't declared in the style of a specific element, the element will revert to the default position: static.

2. Relative Positioning: Positioning relative means, the element will move relatively to its normal/actual position. It allows the use of top, right, bottom, and left properties for adjustment.

Example:

```
.one {
 background-color: powderblue;
 position: relative;
 right: 50px;
}
```

Output:

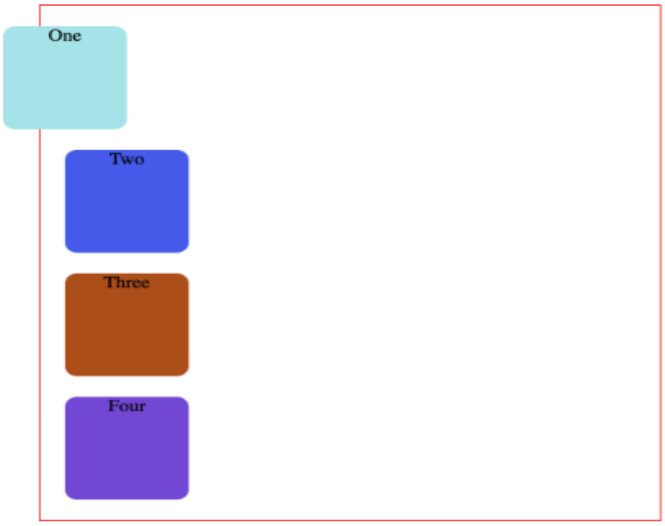


Figure: Relative Positioning

3. Absolute Positioning: Positioning absolute means, the element will move relatively to the nearest positioned element (parent element). If none, it is positioned relative to the initial containing block (usually the <html> element).

Example:

```
.one {
 background-color: powderblue;
 position: absolute;
 top: 50px;
 left: 0;
}
```

Output:

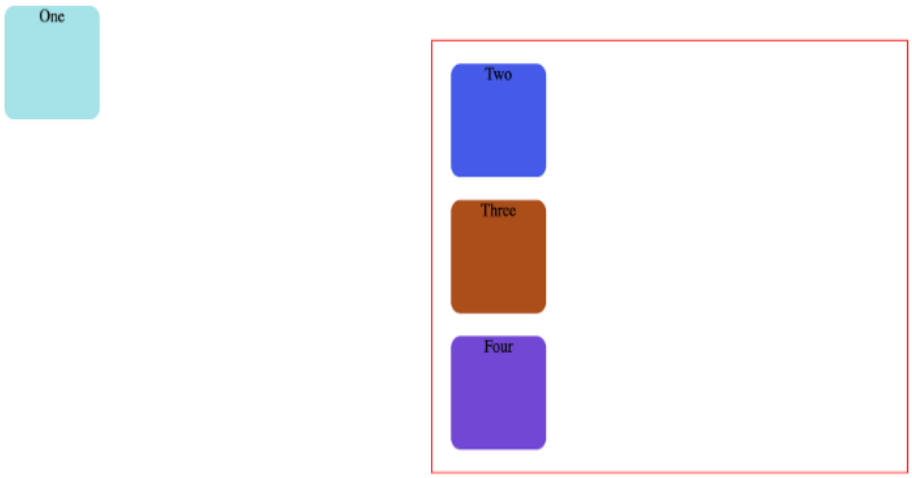


Figure: Absolute Positioning

4. Fixed Positioning: Position fixed means the element will be positioned relative to the browser window. It remains fixed even when the page is scrolled.

Example:

```
.navbar {
 background-color: lightblue;
 position: fixed;
 top: 0;
 width: 100%;
 text-align: center;
}
```

Output:

Figure: Absolute Positioning

5. Sticky Positioning: A sticky positioned element, acts like relative positioning until the element reaches a specified point during scrolling, then it becomes "fixed".

Example:

```
.sticky-navbar {
 position: sticky;
 top: 0;
}
```

CSS z-index: The z-index property specifies the stack order of an element. An element with a higher z-index value is displayed in front of an element with a lower value.

Example:

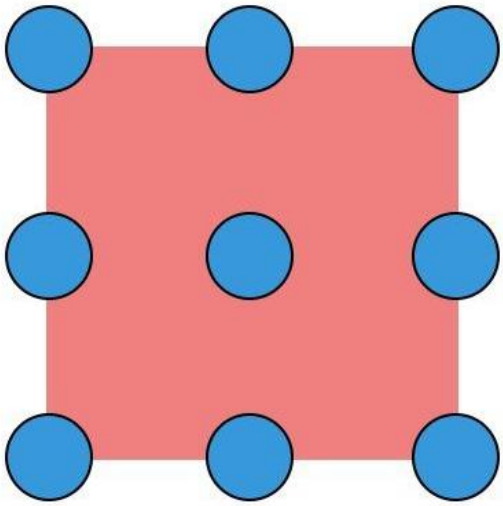
```
#first {
 position: absolute;
 z-index: -1;
 background-color: #192733;
 top: 30px;
 left: 30px;
}
#second {
 position: absolute;
 background-color: #ff5c35;
 top: 10px;
 left: 10px;
}
#third {
 position: absolute;
 z-index: -2;
 background-color: #4fb06d;
 top: 50px;
 left: 50px;
}
```

Output:

Figure: CSS Z-index

Section B: Lab Experiments - Demonstrated by students

Lab Task: Design similar to this picture using the CSS "position" property:



[hints: to make the box rounded, add border-radius: 50% to each box.]

Lab #7: CSS Flexbox and Grid

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 09

Experiment Name: Introduction to CSS Flexbox and Grid.

Theory:

Flexbox: Flexbox is a one-dimensional layout model, primarily designed for arranging elements in a row or a column. It simplifies the creation of complex layouts where elements can dynamically adjust their size and position within a container, depending on available space.

Display flex: This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

Example:

```
.container {
 display: flex;
}
```

Flex Direction: This establishes the main-axis, thus defining the direction flex items are placed in the flex container.

Example:

```
.container {
 display: flex;
 flex-direction: row | row-reverse | column | column-reverse;
}
```

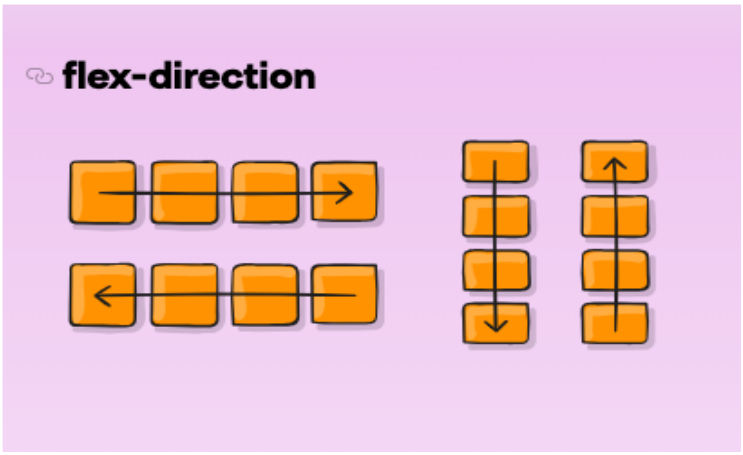


Figure: Flex direction

Justify Content: This defines the alignment along the main axis. It helps distribute extra free space left over when all the flex items on a line are inflexible or flexible but have reached their maximum size.

Example:

```
.container {
    display: flex;
    justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;
}
```

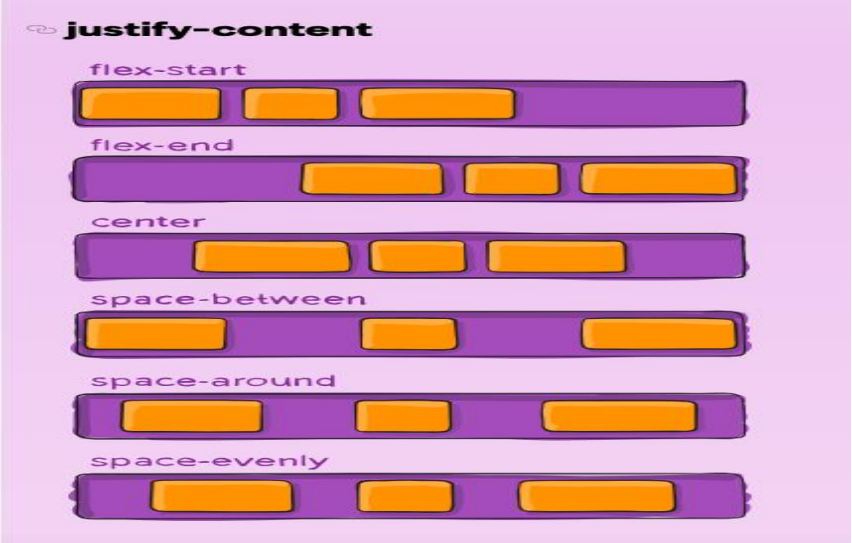


Figure: Justify Flex Contents

Align Items: This defines the default behavior for how flex items are laid out along the cross-axis on the current line.

Example:

```
.container {
    display: flex;
    align-items: stretch | flex-start | flex-end | center | baseline;
}
```

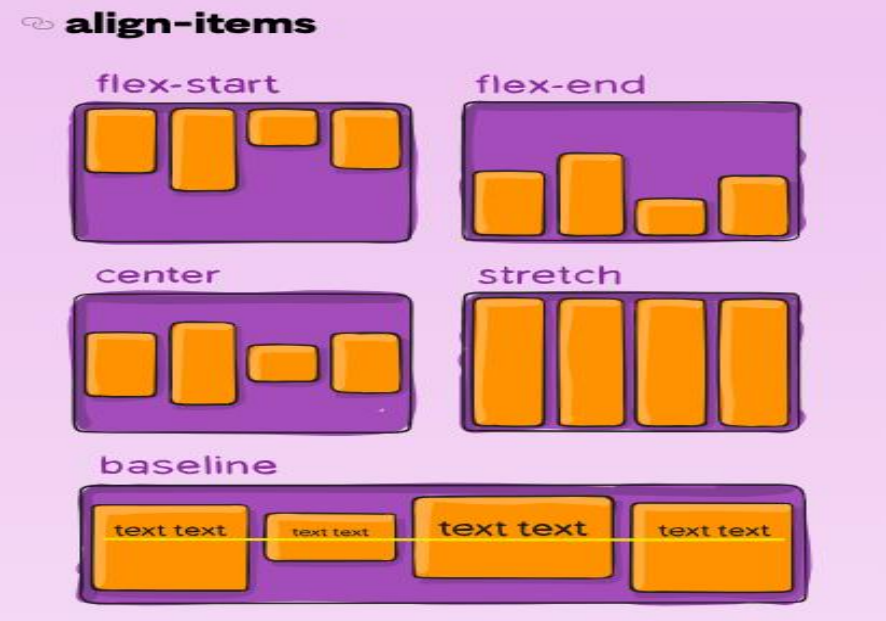


Figure: Align flex items

Gap: The gap property explicitly controls the space between flex items.

Example:

```
.container {
 display: flex;
 gap: 10px;
 gap: 10px 20px; /* row-gap column gap */
 row-gap: 10px;
 column-gap: 20px;
}
```

Source Code:

In index.html:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Document</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="parent">
   <div class="child">Flexbox is a one-dimensional layout model, primarily designed for arranging
elements in a row or a column.
```

```
    </div>
    <div class="child">
      Flexbox is a one-dimensional layout model, primarily designed for
      arranging elements in a row or a column.
    </div>
    <div class="child">
      Flexbox is a one-dimensional layout model, primarily designed for
      arranging elements in a row or a column.
    </div>
    <div class="child">
      Flexbox is a one-dimensional layout model, primarily designed for
      arranging elements in a row or a column.
    </div>
  </div>
 </body>
</html>
```

In style.css:

```
.parent {
 display: flex;
 flex-direction: row;
 gap: 10px;
 padding: 10px;
 height: 600px;
 background-color: tan;
 align-items: center;
 justify-content: center;
}
.child {
 border: 2px solid darkcyan;
 padding: 10px;
 width: 200px;
}
```

Output:

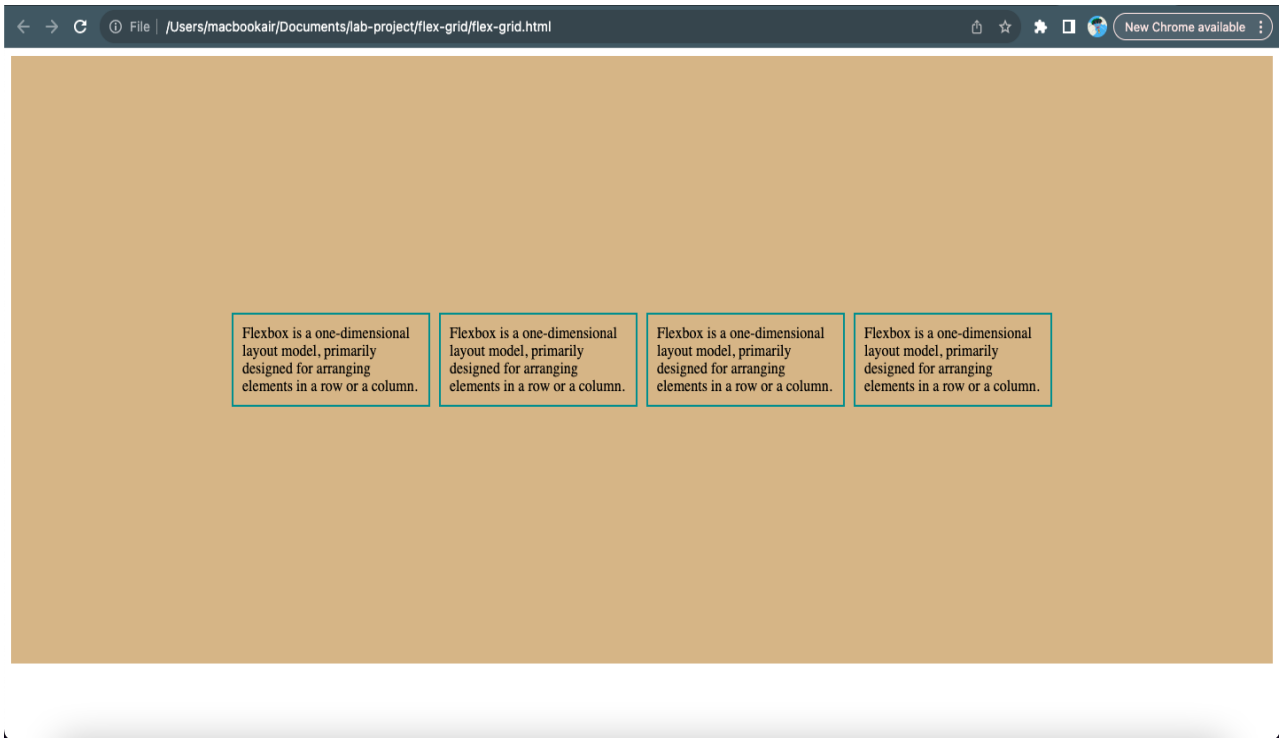


Figure: CSS FlexBox

Grid: CSS Grid is a two-dimensional layout model that allows you to create grid structures consisting of rows and columns. It excels at creating complex, grid-based layouts, such as those found in magazines, newspapers, and responsive web applications.

Example:

```
.container {
 display: grid;
}
```

Grid Columns & Rows: Defines the columns and rows of the grid with a space-separated list of values.

Example:

```
.container {
 display: grid;
 grid-template-columns: 300px 400px 500px;
 /* grid-template-columns: 1fr 1fr 1fr; Each element will take the same size within the container width */
/* grid-template-columns: repeat(3, 1fr); Each element will take the same size within the container width
*/
grid-template-rows: 100px 1fr max-content;
/* grid-template-rows: min-content 1fr min-content; */
}
```

Output:

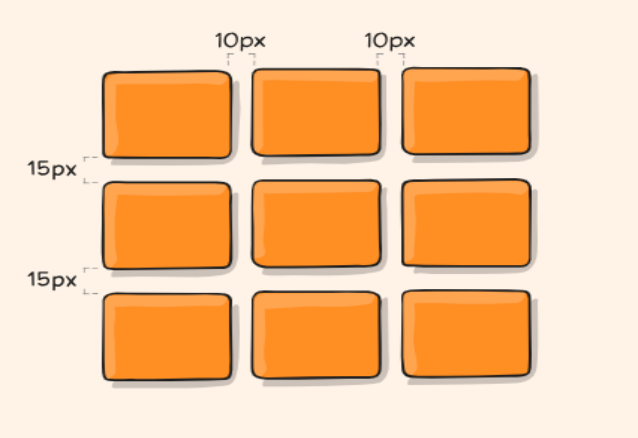


Figure: CSS Grid

Repeat: If your definition contains repeating parts, you can use the repeat() notation to streamline things.

Example:

```
.container {
 display: grid;
 grid-template-columns: repeat(3, 100px);
}
```

Source code:

In index.html file:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Document</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="parent">
   <div class="child">
    Flexbox is a one-dimensional layout model, primarily designed for
    arranging elements in a row or a column.
   </div>
```

```
<div class="child">
    Flexbox is a one-dimensional layout model, primarily designed for arranging elements in a row or a column.
  </div>
  <div class="child">
    Flexbox is a one-dimensional layout model, primarily designed for arranging elements in a row or a column.
  </div>
  <div class="child">
    Flexbox is a one-dimensional layout model, primarily designed for arranging elements in a row or a column.
  </div>
  <div class="child">
    Flexbox is a one-dimensional layout model, primarily designed for arranging elements in a row or a column.
  </div>
  <div class="child">
    Flexbox is a one-dimensional layout model, primarily designed for arranging elements in a row or a column.
  </div>
 </div>
</body>
</html>
```

In style.css file:

```
.parent {
 display: grid;
 grid-template-columns: 1fr 1fr 1fr;
 grid-template-rows: 1fr 1fr;
 gap: 10px;
 padding: 20px;
 height: 600px;
 background-color: tan;
}
.child {
 border: 2px solid darkcyan;
 padding: 10px;
 font-size: 20px;
 width: auto;
}
```

Output:

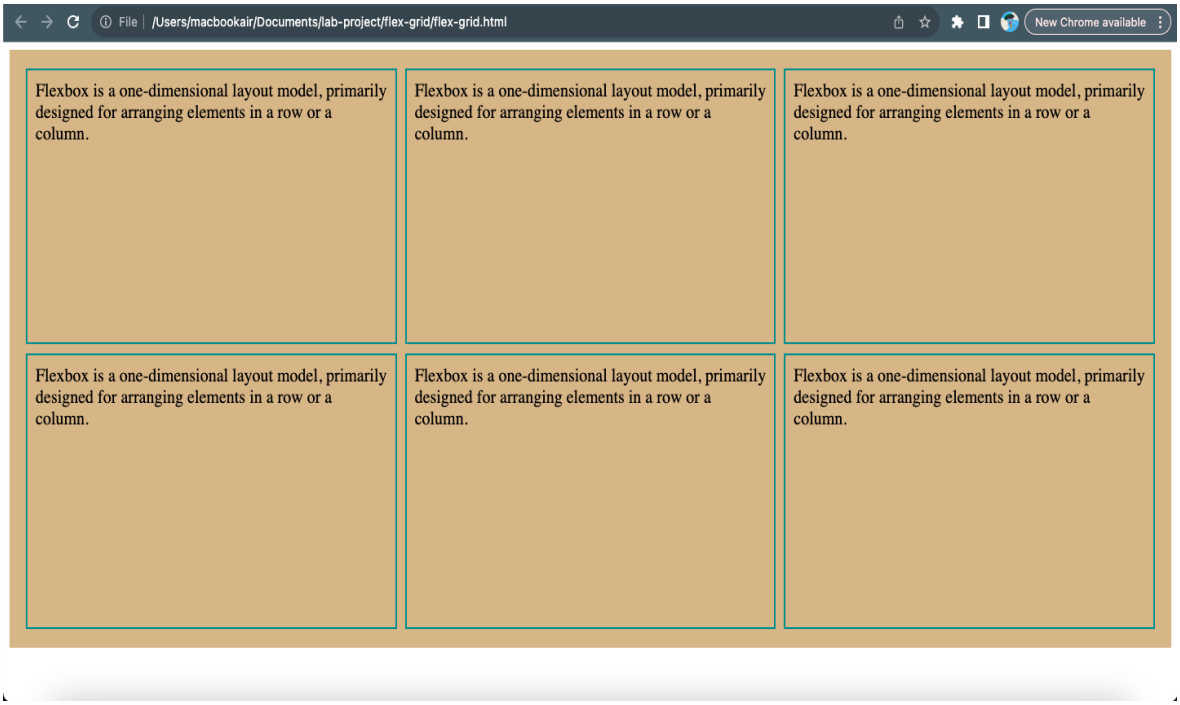


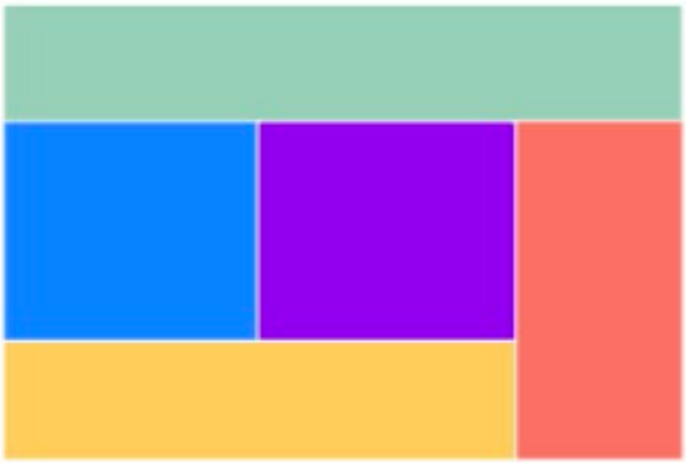
Figure: CSS Grid

Section B: Lab Experiments - Demonstrated by students

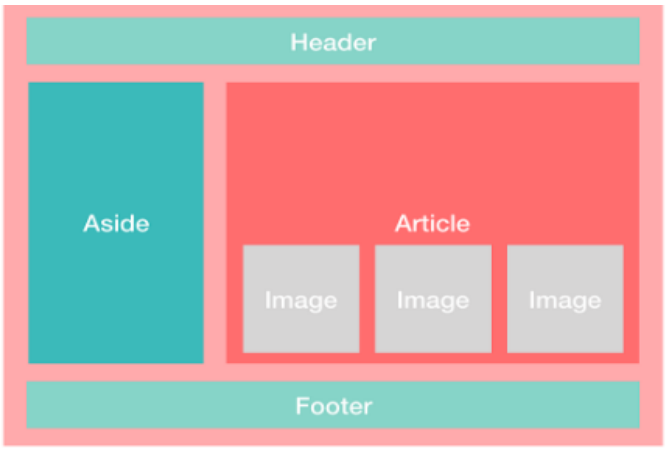
Lab Task - 1: Design web pages similar to the design given below using CSS flex and grid:

1.

2.



Lab Task - 2: Design a web page similar to this:



Lab #8: Introduction to JavaScript

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment: 10

Experiment Name: Introduction to JavaScript.

Theory: JavaScript is a popular programming language that makes web pages interactive and dynamic. It can control HTML elements, respond to user events like clicks, and update content without reloading the page. JavaScript is essential for tasks like form validation, creating animations, and building complex web applications. It runs directly in the browser, making it an important tool for modern web development.

Variables: Variables are used to store and manipulate data. JavaScript follows camelCase naming convention for variables.

Example:

```
var myVariable = "Hello World";
```

Console: The console is used to see the output in the code terminal.

Example:

```
var myVariable = "Hello World";
console.log(myVariable);
```

Data Types: JavaScript has several data types, including String, Number, Boolean, Array, Object, and more.

Example:

```
var name = "John"; // String
var age = 25; // Number
var isStudent = true; // Boolean
var fruits = ["apple", "banana", "orange"]; // Array
var person = {
 firstName: "Jane",
 lastName: "Doe",
}; // Object
```

Math Operators: Perform basic math operations using operators (+, -, *, /).

Example:

```
var sum = 5 + 3; // Addition
var difference = 8 - 4; // Subtraction
```

```
var product = 6 * 2; // Multiplication
var quotient = 10 / 2; // Division
```

Math Methods: JavaScript build-in math methods:

Example:

```
var roundedNumber = Math.round(4.7); // To get the rounded value without the fraction.
var ceilingNumber = Math.ceil(4.3); // To get the max value without the fraction.
var flooringNumber = Math.ceil(4.3); // To get the least value without the fraction.
var randomValue = Math.random(); // Generates a random value between 0 and 1.
var maxValue = Math.max(4, 7, 12, 3); // To get the highest from the given values.
var minValue = Math.max(4, 7, 12, 3); // To get the lowest from the given values.
```

Array: In Javascript, an array is an object that can store multiple values at once within a single variable.

Example:

```
var colors = ["red", "green", "blue"];
console.log(colors.length); // Get the length of the array 'color'
console.log(colors[1]); // Access an element by index (the output is: "green")
console.log(colors.indexOf("green")); // Get the index of an element
colors.push("yellow"); // Add an element to the end of the array
var removedColor = colors.pop(); // Remove and return the last element
var slicedColors = colors.slice(1, 3); // Create a new array by slicing the original
```

[note: JavaScript array index starts from 0]

String: Strings represent text. String can be manipulated in various ways.

Example:

```
var message = "Hello";
var person = "John";
var greeting = message + person; // Concatenate strings
console.log(greeting.toLowerCase()); // Output → hello john
```

Template String: Template strings, also known as template literals, are a feature in JavaScript that makes working with strings more convenient and readable. Instead of using single or double quotes, template strings use backticks (``).

Example:

```
let person = "John";
let greeting = `Hello, ${person}!`;
console.log(greeting); // Outputs: Hello, John!
```

Section B: Lab Experiments - Demonstrated by students

Lab Task - 1: Write a program that declares two variables, '*num1*' and '*num2*' with numeric values of your choice. The program should calculate and display the sum, difference, product, and quotient of these two numbers.

Lab Task - 2: Write a JavaScript program that takes a sentence from the user. Then, display:

a) The length of the sentence.
b) The position of the first occurrence of a specific word (e.g., 'the') in the sentence using indexOf.
c) The sentence is in lowercase format.

Lab Task - 3: Write a JavaScript program that takes your current age as input and calculates the time left until you reach 90 years old in the following format → You have X days, Y weeks, and Z months left. (where X, Y, and Z are replaced with the actual calculated number).

Lab #9: JavaScript Operators, Loops, Functions and Object

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 11

Experiment Name: Introduction to Operators, Loops, and User-Defined Functions in JavaScript.

Theory:

Operators: Operators are used to perform different types of mathematical and logical computations. They come in handy while working with Conditional Statements and Loops:

- '<' → Less than
- '>' → Greater than
- '==' → Is Equal
- '!=' → Is Not Equal
- '<=' → Less than & equal
- '>=' → Greater than & equal

Conditional Statements: JavaScript conditional statements allow to execution of specific blocks of code based on conditions. If the condition is met, a particular block of code will run; otherwise, another block of code will execute based on the condition.

Example:

```
let x = 10;
  if (x > 5) {
        console.log("x is greater than 5"); // If the first condition is true this will execute
  } else if (x == 5) {
        console.log("x is equal to 5");
  } else {
        console.log("x is less than 5");
  }
```

Loops: Looping is essential for efficiently handling repetitive tasks. They execute a block of code repeatedly as long as a specified condition remains true. These loops are powerful tools for automating tasks and streamlining your code.

For Loop: The for loop is commonly used when the number of iterations is known in advance. It consists of three parts: initialization, condition, and iteration expression.

Example:

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

// 'i' means initialization, 'i <=5' indicates condition, and 'i++' is the incremental progress of the loop.

While Loop: The while loop is commonly used when the number of iterations is unknown.

Example:

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

Function: In JavaScript, a function is a block of code that performs a task. It's defined using the function keyword, a name, and parentheses for parameters. The parameters (inputs) are specified within the parentheses, and the function body is enclosed in curly braces.

Example:

```
function sayHello(name) {
  console.log(`Good Morning, ${name}!`);
}

sayHello("Rahim"); // To call the function
```

Function with a Return statement: Functions can have a return statement, which specifies the value that the function will produce. The return statement ends the function's execution and sends the specified value back to the calling code.

Example:

```
function addNumbers(a, b) {
  let sum = a + b
  return sum;
}

const result = addNumbers(10, 20); // Call the function with return value
console.log(result); // 30
```

Object: In JavaScript, an object is allowed to store and organize data using key-value pairs. Each key is a string, and the associated value can be of any data type.

Example:

```
var person = {
  fullname: "Mohammad Nadib Hasan",
  age: 30,
  address: {
    area: "Muradpur",
    district: "Chittagong",
  },
};
```

***** note: Change any property from the given object by: person.age = 35;***

Array of objects: A JavaScript array can also store objects. Arrays of objects are useful for representing collections of related data.

Example:

```
var students = [
  { name: "Alice", age: 22, grade: "A" },
  { name: "Bob", age: 24, grade: "B" },
  { name: "Charlie", age: 23, grade: "C" },
];
```

Section B: Lab Experiments - Demonstrated by students

Lab Task - 1: Write a program to check if a given year is a leap year. The program should take an input year and print a message stating whether the year is a leap year or not.

Lab Task - 2: Write a program that uses a for loop to print the multiplication table for a given number. Allow the user to input the number for which they want to see the multiplication table.

Lab Task - 3: Create a function called *'calculateArea'* that takes the radius of a circle as an argument and returns the area.

Lab Task - 4: Create an array of objects representing your departments. The object should have properties like departmentName, HOD (Head of Department), and an array of students. Again, each student object should include properties like name, studentID, and currentSemester.

Lab #10: DOM Interaction and Manipulation in JS

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 12

Experiment Name: DOM Interaction and Manipulation in JavaScript.

Theory: When a web page loads, the browser reads the HTML code and creates a tree-like structure called the DOM. Each HTML element (such as headings, paragraphs, and buttons) becomes an object in this tree. JavaScript can access and manipulate this DOM tree. It can locate elements, modify their content and style, or even add new elements. This process enables web pages to become interactive. For example, when a button is clicked or a form is filled out, JavaScript can respond by updating the page's display or sending data to the server. In summary, the DOM acts as a bridge between the static HTML content of a web page and the dynamic functionality of JavaScript. It allows JavaScript to read and modify the web page, making websites interactive and responsive.

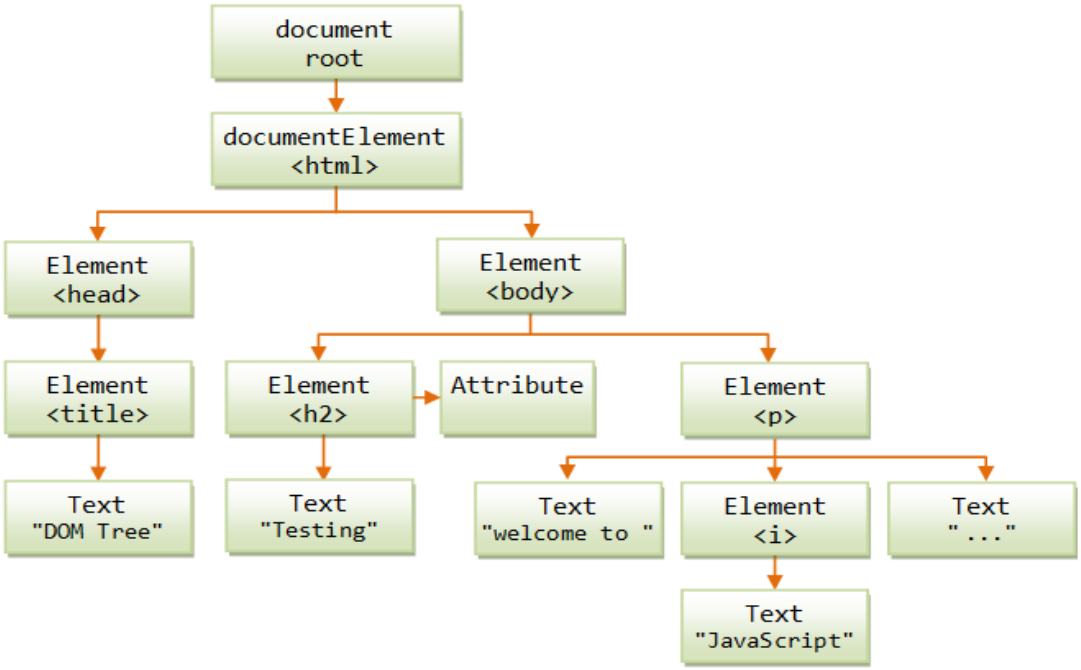


Figure: HTML DOM Tree

Selecting Elements:

- getElementById() → select an element by id.
- getElementsByTagName() → select elements by a tag name.
- getElementsByClassName() → select elements by class name.
- querySelector() → select single elements by a CSS selector.
- querySelectorAll() → select all elements by the same CSS selectors.

Example:

Inside index.html:

```
<body>
  <p id="message">A default paragraph for example…</p>
  <ul>
    <li>Item One</li>
    <li>Item Two</li>
    <li>Item Three</li>
  </ul>
 <div class="container">This is a div with a class.</div>
 <div id="parent">This is a parent component</div>
 <script src="script.js"></script> <!-- to connect the html file with js file -->
</body>
```

Inside script.js:

```
// getElementById
var message = document.getElementById("message");
console.log(message); // It now contains the <p> element that has the value → id=”message”

var elementsByTagName = document.getElementsByTagName("li"); // getElementsByTagName
for (var i = 0; i < elementsByTagName.length; i++) {
  console.log(elementsByTagName[i].textContent);
}

// getElementsByClassName
var elementsByClassName = document.getElementsByClassName("container");
for (var i = 0; i < elementsByClassName.length; i++) {
  console.log(elementsByClassName[i].textContent);
}

// querySelector
var elementBySelector = document.querySelector("#parent");
console.log(elementBySelector.textContent);
```

Manipulating Elements:

- createElement() – create a new element.
- appendChild() – append child node to a specific parent node.
- textContent – get and set the text content of a node.
- innerHTML – get and set the HTML content of an element.
- appendChild() – insert a node after the last child node of a parent node.
- removeChild() – remove child elements of a node.

Example:

```
// using the document.createElement() to create a new <div> element:
var div = document.createElement("div");
// And add an HTML snippet to the div:
div.innerHTML = `<p>Newly created element</p>`;
// To attach the div to the document, you use the appendChild() method:
document.body.appendChild(div);
```

Example:

```
var menu = document.getElementById("menu");
// create new li element
var li = document.createElement("li");
li.innerText = "Home section";
// add it to the ul element
menu.appendChild(li);
```

Element Attributes:

- setAttribute() → set the value of a specified attribute on an element.
- getAttribute() → get the value of an attribute on an element.
- removeAttribute() → remove an attribute from a specified element.
- hasAttribute() → check if an element has a specified attribute or not.

Example:

```
var myInput = document.getElementById("my-input");
myInput.setAttribute("type", "email");
// The input field will turn into this: <input id="my-input" type="email" />
```

Working with Events using addEventListener():

- change → When an HTML element has been changed
- click → When the user clicks an HTML element
- mouseover → When the user moves the mouse over an HTML element
- keydown → When the user pushes a keyboard key

- ❖ submit → When the user submit a form element
- ❖ load → When the browser has finished loading the page

Example:

```
var btn = document.getElementById("btn");
btn.addEventListener("click", function () {
 alert("It is clicked");
});
```

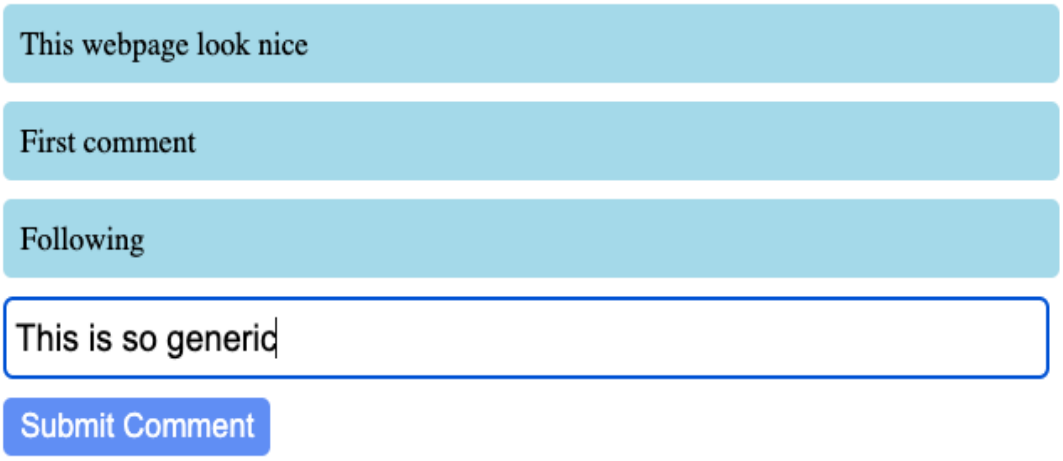
Working with Styles: We can change the style of an element by using javascript dom manipulation. E.g: Element.style.[property_name] = 'value';

Example:

```
var styledElement = document.getElementById("container");
styledElement.style.backgroundColor = "orange";
```

Section B: Lab Experiments - Demonstrated by students

Lab Task: Develop the following interface with HTML and CSS, and then write a JavaScript program to create a comment section where previously added comments are displayed and new comments can be added by clicking a button.



[Hints: Use the input box value as the innerText of the comments body and append the newly created element with the parent].

Lab #11: PHP and Web Server: Introduction and Configuration

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 13

Experiment Name: Introduction to PHP and installation of XAMPP server.

Theory: PHP is a general-purpose scripting language geared towards web development. It was created by Danish-Canadian programmer Rasmus Lerdorf in 1993 and released in 1995. The PHP reference implementation is now produced by the PHP Group. PHP was originally an abbreviation of Personal Home Page, but it now stands for the recursive initialism PHP: Hypertext Preprocessor. PHP is a widely used, open-source scripting language for the backend. PHP files have the extension ".php".

Why we should learn 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 and Open Source.
- PHP is easy to learn and runs efficiently on the server side.

Web Server: A web server is a program that processes the network requests of the users and serves them with files that create web pages. This exchange takes place using the Hypertext Transfer Protocol (HTTP). XAMPP is a free and open-source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages and it is easy to use.

Installation of XAMPP:

Step 1: Download XAMPP software from https://www.apachefriends.org/download.html

Step 2: Double-click on the downloaded .exe file then a pop-up will come up.

Step 3: Keep Clicking on the next button according to instructions.

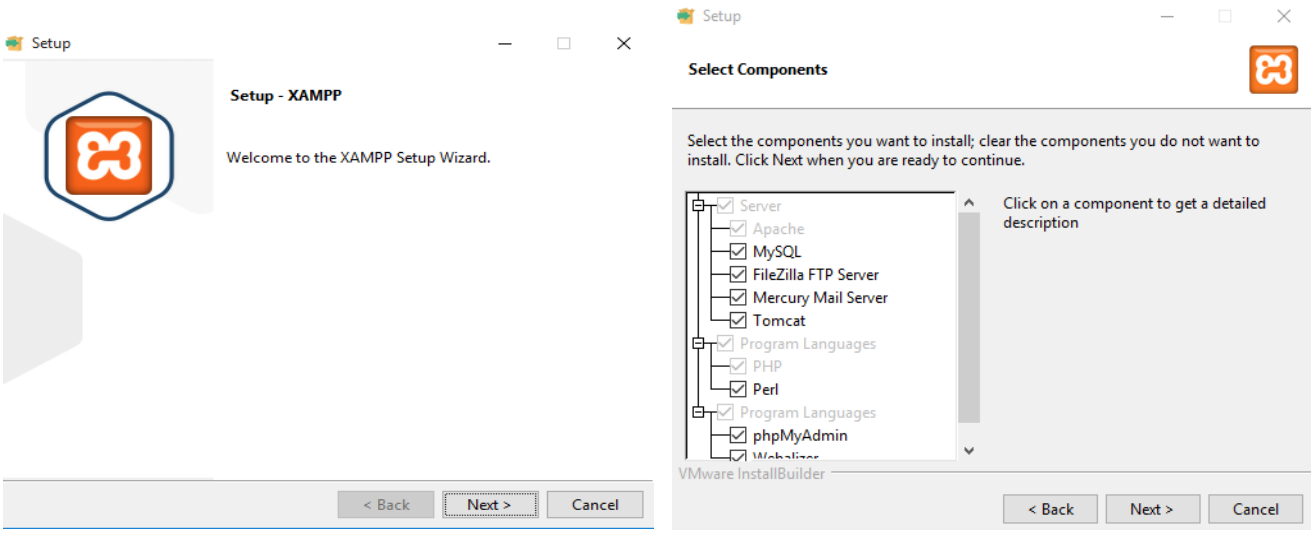


Figure: Installation process of XAMPP server

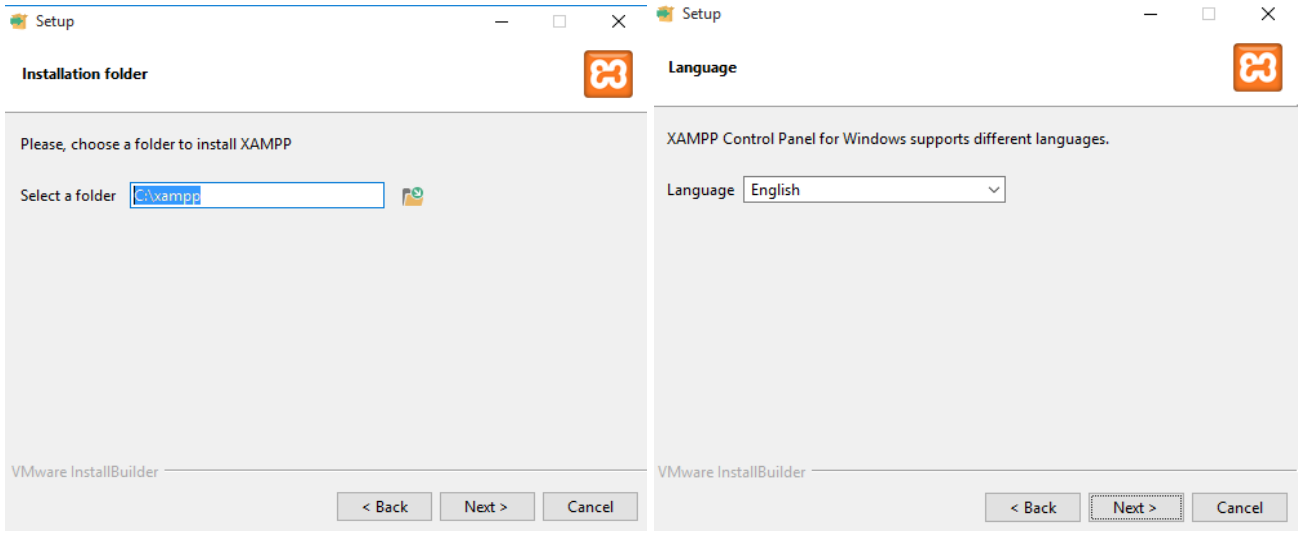


Figure: Installation process of XAMPP server

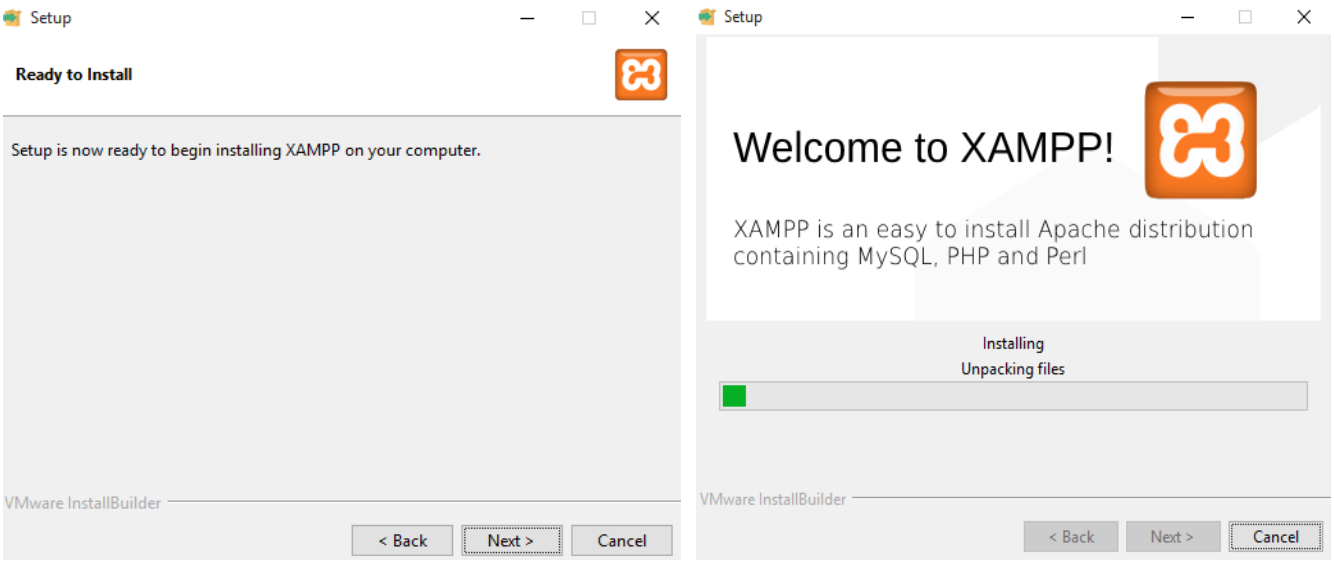


Figure: Installation process of XAMPP server

** After installation open the XAMPP Software and start the Apache

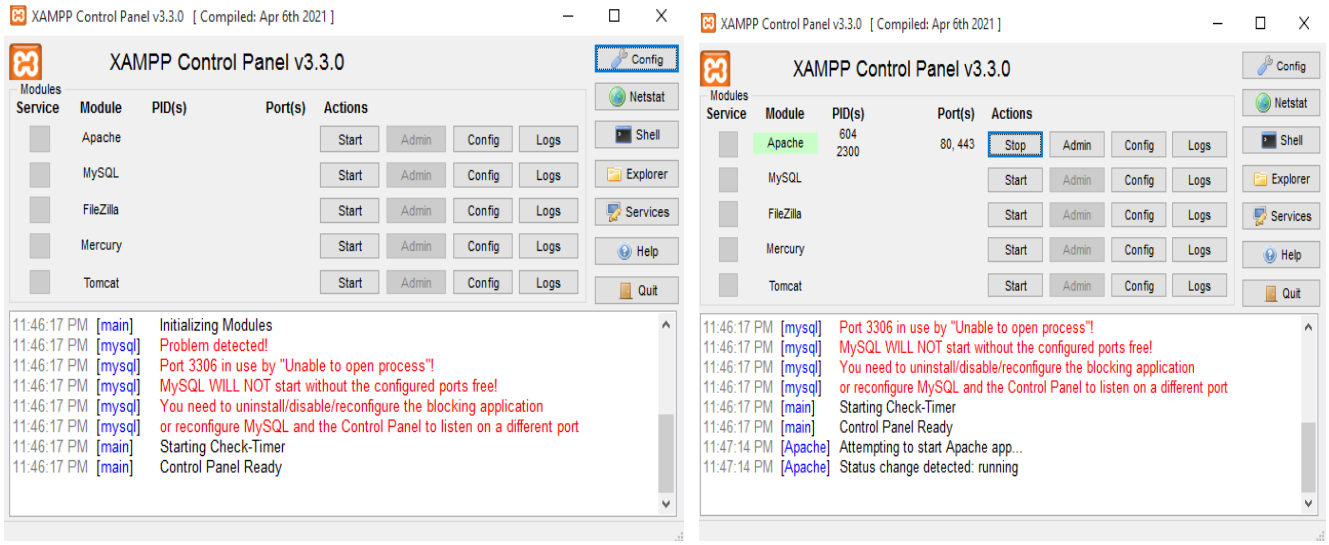


Figure: XAMPP control panel

When the XAMPP server successfully starts to run, the XAMPP Apache will be highlighted…

Then go to your browser and write down in the URL box: http://localhost and the following interface will be presented to you:

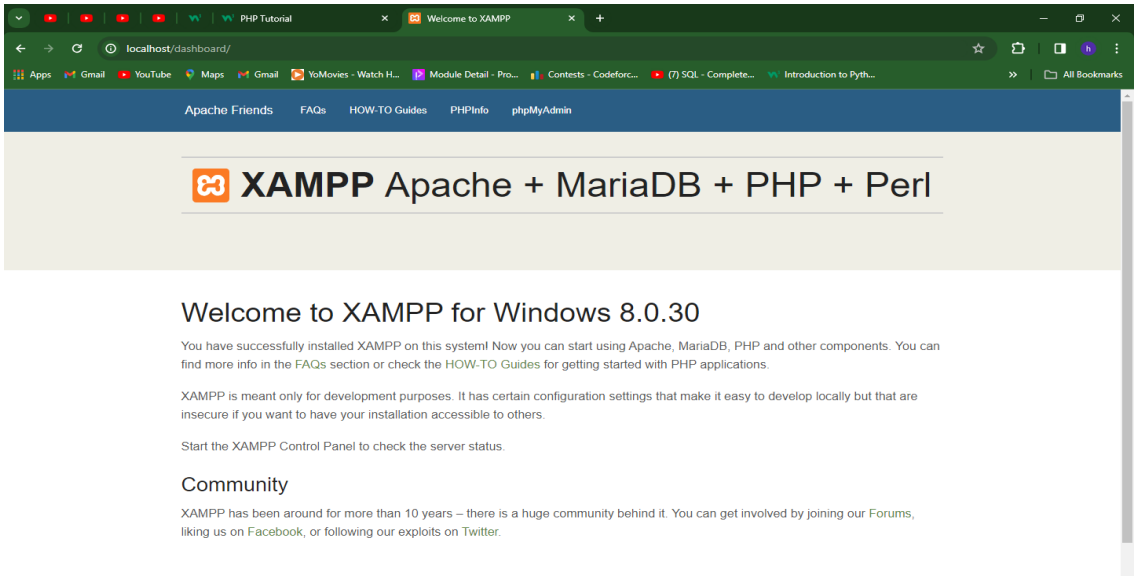


Figure: Dashboard - XAMPP server

[Note: Create a new index.php file and prepare an HTML boilerplate inside it.]

'echo' in PHP: The echo statement outputs text or variables to the browser. It can accept multiple arguments separated by commas:

Example:

```
<!DOCTYPE html>
  <html lang="en">
   <head>
     <title>Introduction to PHP</title>
   </head>
   <body>
     <?php
         echo "Welcome to CCE!";
         // This will print a string on the browser
     ?>
  </body>
</html>
```

Variables: Variables in PHP start with a '$' sign followed by the variable name. They are used to store values:

Example: (inside <?php tag)

```
$name = "John Doe";
$age = 30;
echo "My name is $name and I am $age years old.";
// This will output a formatted string on the browser
```

Strings: Strings in PHP are sequences of characters. They can be enclosed in single quotes ' or double quotes ". Double-quoted strings allow variable interpolation:

Example:

```
$greeting = "Hello";
echo '$greeting, world!'; // Output: $greeting, world!
echo "$greeting, world!"; // Output: Hello, world!
```

Numbers and Arithmetic: PHP supports integers, floats (floating-point numbers), and various arithmetic operations:

Example:

```
$x = 5;
$y = 3.14;
echo $x + $y; // Output: 8.14
echo $x - $y; // Output: 1.86
echo $x * $y; // Output: 15.7
echo $x / $y; // Output: 1.59236
echo $x % $y; // Output: 1 (modulo)
```

Arrays: Arrays in PHP can store multiple values of different types. They are created using square brackets [] or the array() function:

Example:

```
$fruits = ["apple", "banana", "cherry"];
echo $fruits[0]; // Output: apple
//Another way of declaring an array
$ages = array(25, 30, 35);
// adds elements to the end of the array.
array_push($fruits, "Orange");
// adding new item by index
$fruits[3] = “Grapes”;
// unset() will remove an element by its index.
unset($fruits[2]);
```

Associative Arrays: An associative array is a type of array in PHP that uses named keys instead of numeric keys to access and store values.

Example:

```
$person = array(
   "name" => "John Doe",
   "age" => 30,
   "city" => "Chittagong"
);
echo $person["name"]; // Output: John Doe
```

Conditional Statements: Conditional statements in PHP allow to execution of code based on specific conditions. The common types are if, else, elseif, and switch.

- **if** checks if a condition is true and runs the code inside its block.
- **else** runs code if the if condition is false.
- **elseif** checks additional conditions if the first if was false.

Example:

```php
<?php
$year = 2024;
if ($year % 400 == 0) {
   echo "$year is a leap year.";
} elseif ($year % 100 == 0) {
   echo "$year is not a leap year.";
} elseif ($year % 4 == 0) {
   echo "$year is a leap year.";
} else {
   echo "$year is not a leap year.";
}
?>
```

Looping: Like other programming languages, looping is used to execute code repeatedly. Common loop types include for, while, and foreach:

Example:

```php
// for loop in PHP
for ($i = 0; $i < 5; $i++) {
   echo "Iteration $i\n";
}

// while loop in PHP
$i = 1;
while ($i <= 5) {
   echo "The number is: $i <br>";
   $i++;
}

// foreach loop in PHP
$colors = array("red", "green", "blue");
foreach ($colors as $color) {
   echo "The color is: $color\n";
}
```

Function: A user-defined function in PHP is a function that is created by the developer to perform a specific task. It can be used to define a custom block of codes to avoid repetitive tasks.

Example:

```php
function greet($name) {
   return "Hello, $name!";
}

$message = greet("Welcome to CCE Department!");
echo $message;
```

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

Example:

```php
echo date("Y-m-d H:i:s"); // default function to show the current date and time
$date = new DateTime('2024-09-30');
echo $date->format('d-m-y'); // Output: 30-09-2024
```

File Handling: PHP provides functions to read from and write to files. The fopen(), fread(), fwrite(), and fclose() functions are commonly used:

Example:

```php
$file = fopen("example.txt", "w");
fwrite($file, "Hello, Testing File Content!");
fclose($file);

$content = file_get_contents("example.txt");
echo $content;
// The Output will be: Hello, Testing File Content!
```

Error Handling: Error handling in PHP is crucial for developing robust applications. PHP provides several ways to handle errors, including try, catch, and finally blocks for exceptions:

```php
try {
   $result = 10 / 0; // This is a 'Division by Zero' error
} catch (Exception $e) {
   echo "Caught exception: " . $e->getMessage();
} finally {
   echo "This block always executes.";
}
```

Section B: Lab Experiments - Demonstrated by students

Lab Task-1: Create a PHP script that:

- Defines two variables: one for your name (a string) and one for your age (a number).
- Concatenate these two variables in a sentence like: "My name is ABC and I am 25 years old."
- Create an array with three of your favorite colors and print each color on a new line.

Lab Task-2: Write a program in PHP that checks if a given number is even or odd, and outputs "Even" if the number is divisible by 2, or "Odd" otherwise.

Lab Task-3: Write a function in PHP called *'addNumbers'* that takes two numbers as arguments, adds them together, and returns the sum.

Lab Task-4: Create a program in PHP that can write the numbers 1 to 10 into a file named *'numbers.txt'* using a *for loop*. After writing, read the file and check each number. If the number is even, print "X is even", otherwise print "X is odd".

Lab #12: PHP-Form to collect data and store it in a Database

Section A: Tutorial - Demonstrated by Instructor

(30-60 minutes prior to lab Experiments)

Experiment No: 14

Experiment Name: Creation of an HTML Registration and login form with PHP and MySQL using the XAMPP server.

Theory: The backend, or server-side, is the part that users don't see. It involves the server, the application, and the database. Backend development focuses on the functionality, logic, and database operations, ensuring that data is processed correctly and efficiently. PHP is used to power the backends of around 80% of all websites on the internet, and around 40% of them are developed by PHP-based WordPress.

Procedure to make an HTML Registration form:

Step 01: Preparing the Workspace

- Open the file location of XAMPP software.
- Find and open the ***"htdocs"*** folder.
- Create a new folder in a word (If it contains more than one word, used camelCase, PascalCase or use underscore between two words).
- Open the folder in VS Code.

Step 02: Create a new Database

- Open phpMyAdmin from the XAMPP control panel.
- Click New and create a database named ***"user_db"***.
- In the ***"user_db"*** database, create a table called ***"users"*** with the following structure:

Name	Type	Length/Values	Default
id	INT	11	None
fullname	VARCHAR	100	None
email	VARCHAR	255	None
phone	VARCHAR	100	None
password	VARCHAR	100	None

Figure: Database table structure

Step 03: Create a new *"index.php"* file for the registration form:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Registration Form</title>
</head>
<body>
    <h2>Registration Form</h2>
    <form action="register.php" method="POST">
        <label for="fullname">Fullname:</label>
        <input type="text" name="fullname" required><br><br>

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

        <label for="phone">Phone:</label>
        <input type="text" name="phone" required><br><br>

        <label for="password">Password:</label>
        <input type="password" name="password" required><br><br>
        <input type="submit" name="submit" value="Register Now">
</body>
</html>
```

Output:

Figure: User Registration Page

Step 04: Create a Registration PHP Script named *"register.php"*

```php
<?php
$conn = mysqli_connect('localhost', 'root', 'password', 'user_db');

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

if (isset($_POST['submit'])) {
    $fullname = $_POST['fullname'];
    $email = $_POST['email'];
    $phone = $_POST['phone'];
    $password = $_POST['password'];

    $query = "INSERT INTO users (fullname, email, phone, password)
                VALUES ('$fullname', '$email', '$phone', '$password')";

    if (mysqli_query($conn, $query)) {
        echo "New User Registration Successful!";
    } else {
        echo "Error: " . mysqli_error($conn);
    }
}
?>
```

Output: (A new entry on the database)

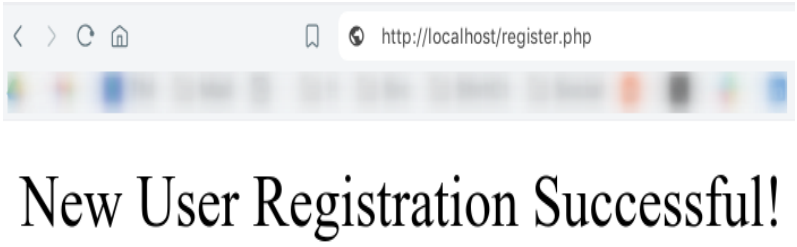


Figure: Successful Message

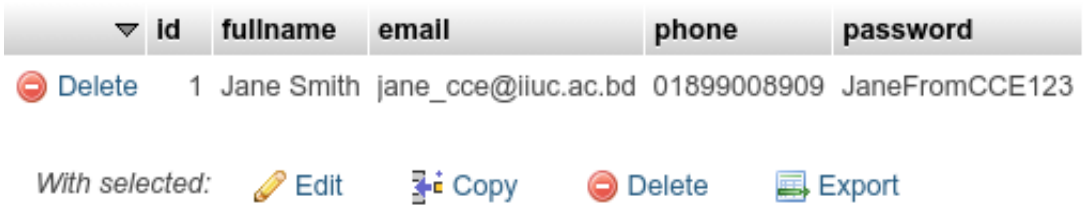


Figure: New entry on Database

Step 05: Create a new *"login.php"* file for the Login page:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Login Form</title>
</head>
<body>
    <h2>Login Form</h2>
    <form action="process_login.php" method="POST">
        <label for="email">Email:</label>
        <input type="email" name="email" required><br><br>

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

        <button type="submit" name="login">Login Now</button>
    </form>
</body>
</html>
```

Step 05: Create a Login PHP Script named *"process_login.php"*

```php
<?php
$conn = mysqli_connect('localhost', 'root', 'password', 'user_db');

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

if (isset($_POST['login'])) {
    $email = $_POST['email'];
    $password = $_POST['password'];

    $query = "SELECT * FROM users WHERE email='$email' AND password='$password'";
    $result = mysqli_query($conn, $query);

    if (mysqli_num_rows($result) == 1) {
        $user = mysqli_fetch_assoc($result);
        echo "Login successful! Welcome, " . $user['fullname'];
    } else {
        echo "Invalid email or password!";
    }
 }
?>
```

Output:

Figure: Login Page Interface

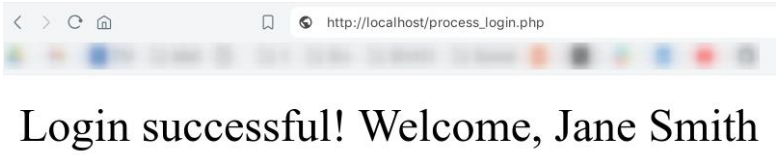


Figure: Successful Login Message

Section B: Lab Experiments - Demonstrated by students

Lab Task: Create a PHP-based system where students can register using their name, email, phone number, student ID, department, and password. After registration, students can log in and view all of their data in an HTML table.

Column Name	Data Type	Length	Extra
id	INT	11	AUTO_INCREMENT, Primary Key
student_name	VARCHAR	100	Not Null
email	VARCHAR	100	Not Null, Unique
phone	VARCHAR	15	Not Null
student_id	VARCHAR	20	Not Null, Unique
department	VARCHAR	100	Not Null
password	VARCHAR	255	Not Null (hashed)

Figure: database 'students' table design