**Assignment 1:-**

**Frontend Assignment Set**

**Module 1 – Foundation**

**THEORY EXERCISE:**

1. What is a HTTP?

* HTTP, which stands for **Hypertext Transfer Protocol**, is the foundation of data communication on the World Wide Web. It is a protocol used for transferring data between a client (like a web browser) and a server. HTTP allows the fetching of resources, such as HTML documents, images, videos, or other files, making it essential for web browsing.

**Key Features of HTTP:**

* **Stateless**: HTTP does not retain information about previous requests. Each request from a client to a server is treated independently.
* **Request-Response Model**:
  + The **client** sends an HTTP request to the server (e.g., to fetch a webpage).
  + The **server** responds with an HTTP response, which may include the requested data (e.g., an HTML file) or an error message.
* **Human-Readable**: HTTP messages (requests and responses) are text-based and easy to understand.

1. What is a Browsers? How they works?

* A **web browser** is a software application that allows users to access, retrieve, and display content from the World Wide Web. Examples include **Google Chrome**, **Mozilla Firefox**, **Safari**, and **Microsoft Edge**. Browsers take input (like a URL) and present web pages, images, videos, or other content by interacting with web servers.

**Working:-**

1. **User Input**:

* The user enters a URL (e.g., https://example.com) into the address bar or clicks a hyperlink.
* Alternatively, the user may perform a search or load a file from local storage.

1. **DNS Resolution**:

* The browser translates the domain name (e.g., example.com) into an IP address using the Domain Name System (DNS).
* This allows the browser to find the server where the website is hosted.

1. **Sending an HTTP/HTTPS Request**:

* The browser establishes a connection to the server (using protocols like **TCP** or **QUIC** for HTTPS).
* It sends an HTTP request to the server, specifying the resource it wants to retrieve.

1. **Receiving the Response**:

* The server processes the request and sends back an HTTP response.
* This response includes a **status code** (e.g., 200 OK, 404 Not Found) and the requested data (e.g., an HTML document, CSS, JavaScript, or images).

1. **Rendering the Content**:

* The browser **parses the HTML** and creates a **DOM (Document Object Model)** tree to structure the webpage.
* It downloads and processes linked resources like CSS (for styling) and JavaScript (for interactivity).

1. **Executing JavaScript**:

* The browser’s **JavaScript engine** (e.g., V8 in Chrome, SpiderMonkey in Firefox) runs any JavaScript code in the page.
* JavaScript can modify the DOM dynamically, enabling interactive and responsive websites.

1. **Continuous Updates**:

* As the user interacts with the webpage (e.g., scrolling, clicking buttons), the browser continuously updates the display.
* It also handles background tasks like prefetching resources or checking for updates.

1. What is Domain Name?

* A **domain name** is the human-readable address used to identify a website or an online resource on the internet. It acts as a simple way for users to access websites without needing to remember complex numerical IP addresses. A domain name typically consists of two main parts:

1. **Second-level domain (SLD)**: This is the part of the domain name that is unique to the website and usually represents the brand, business, or individual, such as "google" in "google.com."
2. **Top-level domain (TLD)**: This is the suffix at the end of the domain name, like ".com," ".org," or ".net." TLDs can also include country-code domains, like ".uk" or ".ca."

For example, in the domain name "[www.example.com](http://www.example.com/)":

* "example" is the **second-level domain (SLD)**
* "com" is the **top-level domain (TLD)**

Together, the full domain name forms a unique address that can be used to locate and access a website.

1. What is hosting?

**Hosting** refers to the service that allows individuals or organizations to store their website files and data on a server, making the website accessible via the internet. In simple terms, web hosting is the process of renting space on a server to store a website's content (such as HTML files, images, databases, etc.) so that users can access it online.

There are different types of hosting services available:

1. **Shared Hosting**: Multiple websites share the same server resources, such as CPU, memory, and bandwidth. This is usually the most affordable option but may offer limited resources.
2. **VPS (Virtual Private Server) Hosting**: A more powerful option than shared hosting, where a physical server is divided into virtual servers. Each user gets dedicated resources, offering better performance and flexibility.
3. **Dedicated Hosting**: In this setup, an entire physical server is dedicated to one website or user. This option provides maximum control, performance, and security but can be more expensive.
4. **Cloud Hosting**: Websites are hosted on a network of servers in the cloud, allowing for scalability and flexibility. Resources can be adjusted as needed, making it suitable for websites with fluctuating traffic.
5. **Managed Hosting**: The hosting provider takes care of the technical aspects of hosting, including server maintenance, software updates, and security, while the user focuses on their website.
6. **WordPress Hosting**: Specifically designed for WordPress websites, this type of hosting comes with pre-configured settings optimized for WordPress performance and security.

In all types of hosting, the key goal is to ensure that websites are available to users around the clock with minimal downtime and fast loading speeds.

**Module 2 – Fundamentals of World Wide Web**

**1. Difference Between Web Designer and Web Developer:**

* **Web Designer**:
  + Focuses on the *look and feel* of a website.
  + Works on the layout, colors, fonts, images, and overall user interface (UI) design.
  + Primarily uses design tools like Photoshop, Illustrator, or web design tools like Figma and Sketch.
  + May also focus on *user experience (UX)* to ensure the site is easy to navigate and visually appealing.
* **Web Developer**:
  + Focuses on the *functionality* of a website, ensuring it works properly.
  + Uses programming languages like HTML, CSS, JavaScript, and backend languages like PHP, Python, Ruby, etc.
  + There are two types of developers:
    - **Frontend Developer**: Works on the user-facing part of the website (using HTML, CSS, JavaScript).
    - **Backend Developer**: Works on the server-side, databases, and business logic (using PHP, Python, Ruby, etc.).

**2. What is W3C?**

* + The **World Wide Web Consortium (W3C)** is an international community that develops open standards to ensure the long-term growth and accessibility of the web. It was founded by Tim Berners-Lee, the inventor of the World Wide Web. W3C creates guidelines and specifications to improve the web’s interoperability, ensuring that different websites, browsers, and technologies can work together seamlessly.

**3. What is a Domain?**

* + A **domain** refers to a unique name that identifies a website on the internet. It’s the address users type in their browser to access a website (e.g., www.example.com). Domains are made up of two main parts:
* **Top-Level Domain (TLD)**: The part after the last dot (e.g., .com, .org, .edu).
* **Second-Level Domain (SLD)**: The name of the website (e.g., example in www.example.com).

**4. What is SEO?**

* + **SEO (Search Engine Optimization)** is the practice of optimizing a website or web page to improve its visibility and ranking on search engines like Google. The goal is to increase organic (non-paid) traffic to the site. SEO includes various techniques, such as:
* **On-page SEO**: Optimizing content, meta tags, keywords, and internal links.
* **Off-page SEO**: Building backlinks and improving external factors that affect search rankings.
* **Technical SEO**: Improving site speed, mobile-friendliness, and indexing.

**5. What is SDLC?**

* + The **Software Development Life Cycle (SDLC)** is a structured approach to software development that defines the stages a software product goes through from inception to retirement. The typical SDLC phases include:

1. **Planning**: Defining project goals, requirements, and resources.
2. **Design**: Creating the architecture and design of the software.
3. **Development**: Writing the actual code for the software.
4. **Testing**: Ensuring the software works as intended through quality assurance (QA).
5. **Deployment**: Releasing the software to users.
6. **Maintenance**: Updating and fixing issues after deployment.

These phases help ensure the software is developed efficiently, meets user needs, and is free of defects.

**Module 3 – Fundamentals of IT**

**1. What is a Program?**

* A **program** is a set of instructions that a computer follows to perform specific tasks. It is written using a programming language and tells the computer how to solve a problem or complete an action, such as calculating numbers, displaying information, or interacting with the user.

**How it Functions:**

* The program works by being executed on a computer's central processing unit (CPU), where it processes the instructions step by step. It can interact with hardware, software, and user input, depending on the task at hand.

**2. What is Programming?**

**Theory Exercise: Key Steps in the Programming Process**

* **Programming** is the process of designing, writing, testing, and maintaining code to create software that performs a specific function.

**Key Steps in the Programming Process:**

1. **Planning**: Understand the problem or task that needs to be solved.
2. **Designing**: Plan how the program will work (algorithm, flowcharts, etc.).
3. **Coding**: Write the code using a programming language.
4. **Testing**: Run the program to find and fix any bugs or issues.
5. **Deployment**: Make the program available for use.
6. **Maintenance**: Update and fix the program over time as needed.

**3. Types of Programming Languages**

**Theory Exercise: High-Level vs. Low-Level Programming Languages**

* **High-Level Languages**: These are closer to human language and abstract away most of the hardware details. Examples include Python, Java, and C#. They are easier to write and understand.
* **Low-Level Languages**: These are closer to machine language and interact more directly with the hardware. Examples include Assembly language and machine code. They are faster but harder to write and maintain.

**4. World Wide Web & How Internet Works**

**Theory Exercise: Roles of the Client and Server in Web Communication**

* **Client**: The client is usually the web browser or application that requests data from a server. It initiates communication with the server by sending a request (e.g., accessing a website).
* **Server**: The server is a system that hosts the data or application. It responds to client requests by sending back the requested data (e.g., a webpage or image).

**5. Network Layers on Client and Server**

**Theory Exercise: Client-Server Communication**

* **Client-Server Communication** involves the client sending a request over the network to the server, which processes the request and sends back the appropriate response. The process often involves multiple network layers, such as:
  + **Application Layer**: The actual data exchange, such as HTTP requests.
  + **Transport Layer**: Ensures reliable data transfer (e.g., TCP).
  + **Network Layer**: Routes data across networks (e.g., IP addresses).
  + **Data Link Layer**: Manages physical communication between devices.

**6. Types of Internet Connections**

**Theory Exercise: Broadband vs. Fiber-Optic Internet**

* **Broadband**: Refers to high-speed internet connections, including DSL, cable, and satellite. The speed is relatively fast, but it can be inconsistent depending on the type of connection and distance from the provider.
* **Fiber-Optic Internet**: Uses light signals through fiber-optic cables, providing very high speeds and reliability with minimal signal loss, but it is often more expensive and less widespread.

**7. Protocols**

**Theory Exercise: Differences Between HTTP and HTTPS**

* **HTTP (Hypertext Transfer Protocol)**: A protocol used for transmitting data over the web without encryption. It is not secure, meaning data sent can be intercepted.
* **HTTPS (Hypertext Transfer Protocol Secure)**: The secure version of HTTP, which encrypts data between the client and server using SSL/TLS, providing protection against interception and eavesdropping.

**8. Application Security**

**Theory Exercise: Role of Encryption in Securing Applications**

* **Encryption** is a process of converting data into a format that cannot be easily understood without the proper decryption key. It is crucial for protecting sensitive data (such as passwords and personal information) from unauthorized access during storage or transmission.

**9. Software Applications and Its Types**

**Theory Exercise: Difference Between System Software and Application Software**

* **System Software**: This includes the operating system (e.g., Windows, macOS) and utility programs that help manage hardware and software resources.
* **Application Software**: These are programs designed to perform specific tasks for the user, like word processors (Microsoft Word), browsers (Google Chrome), or image editors (Photoshop).

**10. Software Architecture**

**Theory Exercise: Significance of Modularity in Software Architecture**

* **Modularity** is the practice of dividing a system into smaller, manageable components (modules). This promotes reusability, easier maintenance, and better collaboration among developers.

**11. Layers in Software Architecture**

**Theory Exercise: Importance of Layers in Software Architecture**

* **Layers** in software architecture separate different aspects of the application, such as data access, business logic, and user interface. This separation allows easier maintenance, scalability, and improves code organization and testing.

**12. Software Environments**

**Theory Exercise: Importance of Development Environment in Software Production**

* A **development environment** includes the tools, libraries, and configurations that support software development. It is essential for coding, testing, and debugging, ensuring that developers can work efficiently and consistently across different stages of production.

**13. Source Code**

**Theory Exercise: Difference Between Source Code and Machine Code**

* **Source Code**: Human-readable code written by programmers in programming languages like Python, C++, or Java.
* **Machine Code**: The low-level binary code that is directly executed by the computer's CPU. It is generated from source code after compilation.

**14. Github and Introductions**

**Theory Exercise: Importance of Version Control in Software Development**

* **Version control** helps manage changes to source code by tracking revisions, which is crucial for collaboration, avoiding conflicts, and ensuring that changes can be reverted if needed. Git is a widely used version control system.

**15. Student Account in Github**

**Theory Exercise: Benefits of Using Github for Students**

* **GitHub** allows students to collaborate on coding projects, track their changes, and store their code in the cloud. It also provides access to open-source projects, enhancing learning and career opportunities.

**16. Types of Software**

**Theory Exercise: Open-Source vs. Proprietary Software**

* **Open-Source Software**: The source code is freely available, and anyone can modify and distribute it (e.g., Linux, Apache).
* **Proprietary Software**: The source code is closed, and usage is restricted by a license (e.g., Microsoft Windows, Adobe Photoshop).

**17. GIT and GITHUB Training**

**Theory Exercise: How GIT Improves Collaboration in Software Development**

* **Git** allows multiple developers to work on the same codebase by managing changes and versions. It supports collaboration by enabling developers to merge changes and resolve conflicts.

**18. Application Software**

**Theory Exercise: Role of Application Software in Businesses**

* **Application Software** plays a crucial role in businesses by improving productivity, streamlining processes, and enabling tasks like communication, data management, and financial tracking (e.g., CRM software, accounting software).

**19. Software Development Process**

**Theory Exercise: Main Stages of the Software Development Process**

1. **Requirement Analysis**
2. **System Design**
3. **Coding**
4. **Testing**
5. **Deployment**
6. **Maintenance**

**20. Software Requirement**

**Theory Exercise: Importance of Requirement Analysis Phase in Software Development**

* **Requirement Analysis** is crucial because it defines the software’s functionality and scope. It helps ensure that the final product meets the user’s needs and expectations.

**21. Software Analysis**

**Theory Exercise: Role of Software Analysis in the Development Process**

* **Software Analysis** involves evaluating the system requirements to ensure that the design and implementation align with the intended objectives and functionality.

**22. System Design**

**Theory Exercise: Key Elements of System Design**

* Key elements include defining the architecture, modules, data flows, interfaces, and hardware requirements needed to meet system requirements.

**23. Software Testing**

**Theory Exercise: Why Software Testing Is Important**

* **Software Testing** ensures the software functions as expected, is free of bugs, and provides a good user experience. It helps identify and fix defects early in the development cycle.

**24. Maintenance**

**Theory Exercise: Types of Software Maintenance**

* **Corrective Maintenance**: Fixing bugs.
* **Adaptive Maintenance**: Updating the software for new environments.
* **Perfective Maintenance**: Improving functionality based on user feedback.
* **Preventive Maintenance**: Ensuring long-term reliability.

**25. Development**

**Theory Exercise: Differences Between Web and Desktop Applications**

* **Web Applications**: Run in a web browser, platform-independent, require an internet connection.
* **Desktop Applications**: Installed on a specific operating system, work offline.

**26** . **Web Application**

**Theory Exercise: Advantages of Web Applications over Desktop Applications**

* Web apps are accessible from any device with a browser, don't require installation, and can be easily updated.

**27. Designing**

**Theory Exercise: Role of UI/UX Design in Application Development**

* **UI/UX Design** ensures that the application is user-friendly, visually appealing, and provides an intuitive experience.

**28. Mobile Application**

**Theory Exercise: Differences Between Native and Hybrid Mobile Apps**

* **Native Apps**: Developed for specific platforms (e.g., iOS, Android) and offer better performance.
* **Hybrid Apps**: Developed with a single codebase for multiple platforms but may have limited functionality.

**29. DFD (Data Flow Diagram)**

**Theory Exercise: Significance of DFDs in System Analysis**

* DFDs help visualize the flow of data within a system, making it easier to analyze and design the system's functionality.

**30. Desktop Application**

**Theory Exercise: Pros and Cons of Desktop Applications Compared to Web Applications**

* **Pros**: More responsive, works offline, deeper integration with the operating system.
* **Cons**: Limited to specific devices, harder to update, requires installation.

**31. Flow Chart**

**Theory Exercise: How Flowcharts Help in Programming and System Design**

* **Flowcharts** help visualize the logical flow of processes in a program or system, making it easier to design and troubleshoot.

**Module 2 – Frontend - HTML**

**HTML Basics**

**Theory Assignment**

**Question 1:** Define HTML. What is the purpose of HTML in web development?

HTML (HyperText Markup Language) is the standard markup language used to create and structure content on the web. It defines the structure of web pages by using a system of tags and attributes. HTML tells web browsers how to display content like text, images, links, tables, forms, and other elements.

Purpose of HTML in Web Development:

* Structure and Layout: HTML provides the basic structure for web pages, enabling developers to define headings, paragraphs, lists, links, and multimedia content.
* Content Organization: It organizes content in a readable format for both users and search engines.
* Web Interactivity: HTML can work with JavaScript and CSS to make web pages interactive and visually appealing.

**Question 2**: Explain the basic structure of an HTML document. Identify the mandatory tags and their purposes.

An HTML document consists of a hierarchical structure that is made up of several elements and tags. The basic structure of an HTML document is as follows:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document Title</title>

</head>

<body>

<!-- Content goes here -->

</body>

</html>

Mandatory Tags and Their Purposes:

1. <!DOCTYPE html>: This declaration defines the document type and version of HTML (HTML5 in this case).
2. <html>: The root element of the document that encapsulates all the content on the web page. It can have a lang attribute to specify the language.
3. <head>: Contains meta-information about the document, such as the character set, viewport settings for responsive design, and the title of the page.
4. <meta>: Provides metadata like the character encoding (<meta charset="UTF-8">) or viewport settings for mobile responsiveness.
5. <title>: Specifies the title of the document that appears in the browser’s title bar or tab.
6. <body>: Contains the visible content of the webpage, such as text, images, videos, and other elements.

**Question 3**: What is the difference between block-level elements and inline elements in HTML? Provide examples of each.

Block-level elements and inline elements define how content is displayed within a web page.

* Block-Level Elements:
  + These elements occupy the full width of their parent container, meaning they start on a new line and stretch out to the full width of the available space.
  + Block-level elements typically contain other block-level or inline elements.
  + Examples: <div>, <h1>, <p>, <ul>, <ol>, <li>, <section>, <article>.
  + Purpose: Used for large structural elements, such as sections, paragraphs, and lists.
* Inline Elements:
  + These elements only take up as much width as necessary to display their content. They do not force content onto a new line.
  + Inline elements cannot contain block-level elements.
  + Examples: <span>, <a>, <img>, <strong>, <em>.
  + Purpose: Used for smaller, stylistic elements within a line of text or for links and images.

**Question 4**: Discuss the role of semantic HTML. Why is it important for accessibility and SEO? Provide examples of semantic elements.

Semantic HTML refers to the use of HTML elements that convey meaning about the content they contain. These elements are designed to be more descriptive and readable for both humans and machines (like search engines and screen readers).

Importance for Accessibility and SEO:

* Accessibility: Semantic tags help screen readers interpret content more accurately for users with visual impairments. They provide context about the structure of the content, making it easier to navigate and understand.
* SEO (Search Engine Optimization): Search engines use semantic HTML to better understand the content of a webpage. Pages that use semantic tags are more likely to rank higher in search results because search engines can properly index the content and understand its structure.

Examples of Semantic HTML Elements:

1. <header>: Represents the introductory section of a page or a section of content, usually containing logos, navigation, and other introductory elements.
2. <footer>: Represents the footer of a page or section, usually containing copyright information, links, or contact details.
3. <article>: Defines a piece of content that can stand alone, such as a blog post, news article, or a product description.
4. <section>: Represents a thematic grouping of content, typically with a heading. It can be used for logical sections of a page (e.g., a "contact" section).
5. <nav>: Represents a section of navigation links.
6. <aside>: Represents content that is tangentially related to the content around it, such as a sidebar or related links.
7. <main>: Represents the primary content of a document, typically excluding navigation, footers, and sidebars.

Using semantic elements helps structure the content logically and provides a clearer understanding of the webpage to both users and search engines.

**HTML Forms**

**Question 1:** What are HTML forms used for? Describe the purpose of the input, textarea, select, and button elements.

HTML Forms are used to collect user input from web pages and send it to a server for processing. They are crucial for interactive websites where users need to submit data, such as in login forms, registration forms, contact forms, and surveys.

Here are the purposes of the different form elements:

1. <input> element:
   * The <input> element is one of the most commonly used elements in HTML forms. It allows users to enter data in various formats, such as text, numbers, dates, passwords, and more.
   * Attributes: The type attribute determines the kind of input (e.g., type="text", type="password", type="email").
   * Example: <input type="text" name="username" /> creates a text input for the user to type their username.
2. <textarea> element:
   * The <textarea> element is used for multi-line text input, allowing users to input longer or more complex data, such as comments or messages.
   * Attributes: rows and cols control the size of the textarea box, specifying how many lines and columns the user can see when typing.
   * Example: <textarea name="message" rows="4" cols="50"></textarea> creates a multi-line input field for user feedback.
3. <select> element:
   * The <select> element is used to create a dropdown list of options for users to choose from. It is typically used for choices like selecting a country, category, or preference.
   * Attributes: The <select> element contains <option> elements that represent the available choices.
   * Example:
   * <select name="country">
   * <option value="usa">United States</option>
   * <option value="canada">Canada</option>
   * </select>
   * This creates a dropdown list with "United States" and "Canada" as options.
4. <button> element:
   * The <button> element is used to trigger actions in forms, such as submitting the form or performing a custom action using JavaScript.
   * Attributes: The type attribute defines the behavior (e.g., type="submit" for form submission or type="button" for general-purpose buttons).
   * Example: <button type="submit">Submit</button> creates a button that submits the form when clicked.

**Question 2:** Explain the difference between the GET and POST methods in form submission. When should each be used?

GET and POST are two methods used for sending data from a form to a server.

* GET Method:
  + Purpose: The GET method appends form data to the URL as query parameters. It is used when the data being submitted is not sensitive and does not need to be hidden.
  + Characteristics:
    - Data is visible in the URL (e.g., example.com/search?query=dog).
    - It has length limitations (URLs can only hold a limited amount of data).
    - It is suitable for non-sensitive data, such as search queries or filtering data.
  + When to use: Use GET for retrieving data or when submitting non-sensitive data that can be included in the URL.
  + Example: A search form where the user types a query and the results are displayed based on the query (e.g., search results for "dog").
* POST Method:
  + Purpose: The POST method sends form data in the body of the HTTP request, rather than appending it to the URL. This method is used when submitting sensitive data, or when large amounts of data need to be sent.
  + Characteristics:
    - Data is not visible in the URL, making it more secure for transmitting sensitive information (e.g., passwords).
    - There is no significant size limit for data submission.
    - POST is used for submitting data to be processed (e.g., login forms, registration forms, file uploads).
  + When to use: Use POST for submitting sensitive data (like passwords) or when submitting large amounts of data (e.g., a file upload).
  + Example: A login form that submits a username and password securely.

**Question 3:** What is the purpose of the label element in a form, and how does it improve accessibility?

The <label> element is used to define a label for a form control (such as an <input>, <textarea>, or <select>). It associates text with a specific form field, providing a clearer user experience, especially for screen readers or users with disabilities.

Purpose of the <label> element:

1. Clarifies form field purpose: The label describes what data should be entered in the form field (e.g., "Username", "Email", "Password").
2. Improves accessibility: Associating labels with form controls helps screen readers read out the label when a user navigates through form fields, improving the accessibility for visually impaired users.
3. Clickable label: When the <label> is clicked, it focuses on the associated form control, making the form more user-friendly. For example, clicking a label like "Email" will focus on the corresponding <input> field.

How it improves accessibility:

* For screen readers: The <label> provides additional context for users who rely on assistive technologies, making it clear which input field corresponds to which label. This is essential for users with visual impairments.
* For form navigation: Users can click on the label to activate the corresponding input field, making it easier to interact with forms, especially on mobile devices where input fields can be hard to target.

Example:

<label for="username">Username:</label>

<input type="text" id="username" name="username" />

* In this example, clicking on the "Username" label will focus on the associated <input> field, improving usability for all users. The for attribute in the <label> tag must match the id attribute of the corresponding input field to establish the association.

**HTML Tables**

**Question 1:** Explain the structure of an HTML table and the purpose of each of the following elements: <table>, <tr>, <th>, <td>, and <thead>.

An HTML table is a structure used to display data in a grid format, arranged in rows and columns. It is commonly used for representing tabular data, such as schedules, product lists, or reports.

The basic structure of an HTML table consists of several key elements:

1. <table>:
   * Purpose: The <table> element is the container for the entire table structure. It holds all the other table-related elements (such as rows, headers, and data cells).
   * Example:
   * <table>
   * <!-- Table content goes here -->
   * </table>
2. <tr> (Table Row):
   * Purpose: The <tr> element defines a row in the table. It groups together <th> (header cells) and <td> (data cells) elements within a row.
   * Example:
   * <tr>
   * <td>Row 1, Column 1</td>
   * <td>Row 1, Column 2</td>
   * </tr>
3. <th> (Table Header Cell):
   * Purpose: The <th> element is used for header cells in the table. These cells are usually bolded and centered by default, and they represent the heading of each column or row. Text in <th> cells is typically interpreted as column or row labels.
   * Example:
   * <th>Column Header</th>
4. <td> (Table Data Cell):
   * Purpose: The <td> element is used for normal data cells in the table. It contains the actual data or content of the table that corresponds to the row and column intersection.
   * Example:
   * <td>Row 1, Column 1 Data</td>
5. <thead> (Table Header Group):
   * Purpose: The <thead> element is used to group header rows in the table. It is used to separate the header section of a table from the body section (<tbody>) and foot section (<tfoot>). This allows for easier styling and management, especially in tables with multiple rows.
   * Example:
   * <thead>
   * <tr>
   * <th>Header 1</th>
   * <th>Header 2</th>
   * </tr>
   * </thead>

Full Example of a simple table:

<table>

<thead>

<tr>

<th>Header 1</th>

<th>Header 2</th>

</tr>

</thead>

<tbody>

<tr>

<td>Row 1, Column 1</td>

<td>Row 1, Column 2</td>

</tr>

</tbody>

</table>

**Question 2:** What is the difference between colspan and rowspan in tables? Provide examples.

* colspan: The colspan attribute is used in table cells (<th> or <td>) to make the cell span across multiple columns. It specifies the number of columns a cell should occupy.

Example of colspan:

<table>

<tr>

<td colspan="2">This cell spans 2 columns</td>

</tr>

<tr>

<td>Row 2, Column 1</td>

<td>Row 2, Column 2</td>

</tr>

</table>

In this example, the first cell in the first row will span across two columns.

* rowspan: The rowspan attribute is used in table cells (<th> or <td>) to make the cell span across multiple rows. It specifies the number of rows a cell should occupy.

Example of rowspan:

<table>

<tr>

<td rowspan="2">This cell spans 2 rows</td>

<td>Row 1, Column 2</td>

</tr>

<tr>

<td>Row 2, Column 2</td>

</tr>

</table>

In this example, the first cell in the first column spans across two rows.

Summary:

* colspan: Spans across multiple columns.
* rowspan: Spans across multiple rows.

**Question 3:** Why should tables be used sparingly for layout purposes? What is a better alternative?

Using tables for layout purposes is not recommended because it was historically a workaround for structuring web pages before the development of more powerful and flexible layout techniques like CSS. However, tables are designed specifically for presenting tabular data and should not be misused for general layout purposes.

Reasons to avoid using tables for layout:

1. Accessibility: Using tables for layout can make it harder for screen readers and assistive technologies to interpret content correctly. Tables should be used for tabular data, not for structuring non-data content.
2. Maintainability: Table-based layouts are more difficult to maintain, especially in large websites. Changes to the layout often require complex adjustments to the table structure.
3. Flexibility: Tables are rigid and lack the flexibility offered by CSS for creating responsive, adaptive, and more dynamic layouts across various screen sizes.
4. Semantic HTML: Using tables for non-tabular content violates the principle of semantic HTML, which focuses on using the correct HTML element for the intended purpose (e.g., using <div> or <section> for layout, and <table> only for data).

Better Alternatives (CSS Layout Techniques):

* CSS Grid Layout: A powerful layout system that allows for complex two-dimensional layouts with rows and columns.
* CSS Flexbox: A flexible layout model designed to arrange elements in one-dimensional spaces (either horizontally or vertically).
* Responsive Design: CSS media queries allow content to adapt to different screen sizes, improving user experience across devices.

Example of CSS Flexbox Layout:

<div class="flex-container">

<div class="item">Item 1</div>

<div class="item">Item 2</div>

<div class="item">Item 3</div>

</div>

<style>

.flex-container {

display: flex;

justify-content: space-between;

}

.item {

background-color: lightgray;

padding: 10px;

}

</style>

In this example, the flexbox layout is used to evenly space out the items, creating a modern, flexible layout.

In conclusion, CSS provides far better tools for creating web layouts compared to tables, offering more flexibility, ease of maintenance, and improved accessibility.

**Module 5 – Frontend – HTML5**

**Question 1:** Difference between HTML and HTML5

HTML (Hypertext Markup Language) is the standard language used to create and design webpages. HTML has undergone several updates over the years, with HTML5 being the latest version. Here's a comparison between HTML and HTML5:

| Feature | HTML | HTML5 |
| --- | --- | --- |
| Doctype Declaration | <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" ...> | <!DOCTYPE html> (Simplified) |
| Multimedia Support | No built-in support for audio or video elements. External plugins like Flash needed. | Native support for <audio> and <video> tags to embed multimedia without the need for plugins. |
| Semantic Elements | Limited use of semantic tags. Commonly uses <div> and <span> for layout and structure. | Introduces semantic tags like <article>, <section>, <nav>, <header>, <footer>, and <figure>, improving readability and accessibility. |
| Form Elements | Limited form input types (e.g., <input type="text">). | New input types for forms such as email, url, date, range, color, and more, allowing better validation and functionality. |
| API Support | No support for native APIs. JavaScript often used for interactivity. | HTML5 introduces various APIs like the Canvas API, Geolocation API, Web Storage API, Web Workers, and more for richer, interactive web experiences. |
| Graphics | No native support for drawing or dynamic graphics. | <canvas> element added for drawing graphics, animations, and dynamic visualizations directly in the browser using JavaScript. |
| Offline Web Apps | Web pages required constant internet connection to work. | HTML5 introduced Web Storage and Application Cache, enabling offline functionality for web apps. |
| Character Encoding | Often used charset="ISO-8859-1". | By default, HTML5 uses UTF-8 character encoding for better international character support. |
| Deprecated Elements | Tags like <font>, <center>, <marquee>, and others were commonly used. | These tags are removed in HTML5, encouraging cleaner, more semantic code using CSS for styling and positioning. |

In summary, HTML5 is a more advanced, feature-rich, and semantic version of HTML, with better support for multimedia, interactivity, and modern web development techniques.

**Question 2:** What are the additional tags used in HTML5?

HTML5 introduced several new tags that offer better structure, semantics, and functionality for modern web development. Here are some of the key additional tags:

1. <article>:
   * Represents a self-contained, independent piece of content, such as a news article, blog post, or user-generated content.
   * Example:
   * <article>
   * <h2>HTML5: The Future of Web Development</h2>
   * <p>HTML5 introduces many new features...</p>
   * </article>
2. <section>:
   * Represents a section of content within a document, often used for grouping related content such as chapters, headers, or sections in a report.
   * Example:
   * <section>
   * <h2>Introduction</h2>
   * <p>HTML5 enhances the way we build websites...</p>
   * </section>
3. <nav>:
   * Used to define a section of navigation links, such as a menu or a set of links to important pages.
   * Example:
   * <nav>
   * <ul>
   * <li><a href="#home">Home</a></li>
   * <li><a href="#about">About</a></li>
   * </ul>
   * </nav>
4. <header>:
   * Represents introductory content or a group of navigational links for the webpage or section. Often contains logos, navigation menus, or page titles.
   * Example:
   * <header>
   * <h1>My Website</h1>
   * <nav>...</nav>
   * </header>
5. <footer>:
   * Defines footer content for a webpage or section, usually containing copyright information, contact details, or links.
   * Example:
   * <footer>
   * <p>© 2024 My Website</p>
   * </footer>
6. <figure>:
   * Used to group media content (images, videos, etc.) and their captions, typically paired with the <figcaption> tag.
   * Example:
   * <figure>
   * <img src="html5\_logo.png" alt="HTML5 Logo">
   * <figcaption>HTML5 Logo</figcaption>
   * </figure>
7. <figcaption>:
   * Provides a caption or description for content inside a <figure>.
   * Example:
   * <figcaption>This is an image of the HTML5 logo.</figcaption>
8. <mark>:
   * Represents highlighted or marked text, typically used to highlight search results or important terms.
   * Example:
   * <p>HTML5 introduced many new features, such as <mark>video</mark> and <mark>audio</mark> support.</p>
9. <progress>:
   * Represents the completion progress of a task, often used in forms or as a visual indicator for ongoing processes.
   * Example:
   * <progress value="50" max="100">50%</progress>
10. <meter>:
    * Represents a scalar measurement within a known range (e.g., disk usage, temperature, or completion percentage).
    * Example:
    * <meter value="0.6" min="0" max="1">60%</meter>
11. <canvas>:
    * Provides an area for drawing graphics via JavaScript. It can be used for creating dynamic graphics, games, animations, and more.
    * Example:
    * <canvas id="myCanvas" width="200" height="100"></canvas>
12. <video>:
    * Used for embedding video content directly into a webpage without relying on third-party plugins like Flash.
    * Example:
    * <video width="320" height="240" controls>
    * <source src="movie.mp4" type="video/mp4">
    * Your browser does not support the video tag.
    * </video>
13. <audio>:
    * Used for embedding audio content, such as music or sound files, into a webpage.
    * Example:
    * <audio controls>
    * <source src="audio.mp3" type="audio/mp3">
    * Your browser does not support the audio element.
    * </audio>
14. <source>:
    * Specifies multiple media resources for elements like <video> or <audio>, allowing browsers to choose the appropriate format.
    * Example:
    * <video controls>
    * <source src="movie.ogg" type="video/ogg">
    * <source src="movie.mp4" type="video/mp4">
    * </video>
15. <details> and <summary>:
    * <details> is used to create a disclosure widget from which the user can view or hide content. <summary> defines a visible heading for the <details> element.
    * Example:
    * <details>
    * <summary>More info</summary>
    * <p>Here is some additional information.</p>
    * </details>

These are just a few of the many new elements introduced in HTML5. These tags help make web development more semantic, structured, and accessible while providing better support for modern media, interactive content, and application features.