HTML Basics

Basic working principle of Internet

The basic working principle of the Internet involves the communication and transfer of data between devices over a global network using standardized protocols. Here's a technical explanation of how the Internet operates:

1. Data Transmission (Packet Switching)

The Internet uses a method called **packet switching** to transmit data. Large chunks of data (such as a file or web page) are broken down into smaller units known as **packets**. Each packet contains a portion of the data and routing information, such as the source and destination IP addresses.

These packets are sent independently across the network and may take different routes to their destination. Once they reach the recipient, the packets are reassembled in the correct order to reconstruct the original data.

2. Internet Protocol (IP)

The **Internet Protocol (IP)** is responsible for addressing and routing packets between devices across networks. Every device connected to the Internet is assigned a unique IP address, which acts as its identifier. This address allows routers to direct packets toward the correct destination, ensuring that data reaches the appropriate device.

There are two versions of IP in use today:

- **IPv4** (Internet Protocol version 4): Uses a 32-bit address space (e.g., 192.168.0.1).
- **IPv6** (Internet Protocol version 6): Uses a 128-bit address space to accommodate the growing number of devices.

3. Transmission Control Protocol (TCP)

The **Transmission Control Protocol (TCP)** works alongside IP and ensures reliable delivery of data. TCP breaks down the data into packets, ensures packets are delivered correctly, and reassembles them at the destination. It also handles error checking and retransmits lost or corrupted packets. It provides the reliable connection needed for activities such as browsing websites and sending emails.

- **Three-Way Handshake**: TCP establishes a connection between the sender and receiver through a process called the **three-way handshake**, where both parties agree to communicate and synchronize their state before data transfer begins.

4. Domain Name System (DNS)

The **Domain Name System (DNS)** is responsible for converting human-readable domain names (e.g., www.example.com) into IP addresses. When you type a URL into your browser, the DNS server looks up the corresponding IP address for that domain, allowing your device to connect to the correct web server.

5. Routing and Switching

The Internet consists of a network of interconnected devices and networks. **Routers** are responsible for forwarding packets across different networks toward their destination. They examine the destination IP address in each packet and use routing tables to determine the best path to forward the packet.

Switches operate within local networks (such as home or office networks) and forward data packets between devices within the same network.

6. Application Layer Protocols

Several **application layer protocols** provide the interfaces through which services are delivered over the Internet. Some key protocols include:

- **Hypertext Transfer Protocol (HTTP)**: Used for web browsing and accessing websites.
- **File Transfer Protocol (FTP)**: Used for transferring files.
- **Simple Mail Transfer Protocol (SMTP)**: Used for sending emails.
- **Post Office Protocol (POP)** and **Internet Message Access Protocol (IMAP)**: Used for retrieving emails.

7. Physical Layer (Infrastructure)

The **physical infrastructure** of the Internet consists of physical cables (fiber optics, copper wires), wireless communication systems (Wi-Fi, LTE, 5G), and satellite links. These physical devices and technologies provide the backbone through which data is transmitted globally.

Internet Service Providers (ISPs) maintain the infrastructure and offer Internet access to consumers, businesses, and data centers.

8. Security and Encryption

To ensure secure communication, protocols like **Secure Sockets Layer (SSL)** or **Transport Layer Security (TLS)** are used to encrypt data during transmission. This prevents unauthorized parties from intercepting and accessing sensitive information, such as passwords and financial data, during communication.

In a technical context, the Internet functions as a global, decentralized network that allows devices to communicate using standardized protocols like IP, TCP, and DNS. Data is transmitted in packets across various networks, with routers and switches directing the flow of information. Application layer protocols enable various services (e.g., browsing, email), and encryption ensures secure communication between devices. The entire system is built on an underlying physical infrastructure, including cables, wireless signals, and satellites, managed by ISPs.

HTTP Protocol, significance & HTTP Stack:

HTTP Protocol (Hypertext Transfer Protocol)

HTTP stands for **Hypertext Transfer Protocol**, and it is the fundamental protocol used for communication between clients (such as web browsers) and web servers on the World Wide Web (WWW). HTTP defines the structure of requests and responses for fetching web resources, such as HTML pages, images, videos, and other content, from a server to a client. HTTP is an **application layer** protocol in the **OSI (Open Systems Interconnection) model**, and it works on top of lower-level protocols like **Transmission Control Protocol (TCP)** for reliable data transfer.

Key Features of HTTP:

1. **Stateless**: HTTP is a stateless protocol, meaning each HTTP request is independent and does not retain information about previous requests. After a request is processed, the connection is closed, and the server does not remember any previous interactions. This is useful for scalability and simplifies server design, but it also means additional mechanisms like cookies or sessions are needed to maintain state.
2. **Request-Response Model**: HTTP follows a simple request-response model where:
   - **Client (Browser)** sends an HTTP request to the server.
   - **Server** processes the request and sends back an HTTP response, which includes the requested data or a status message (e.g., 404 for "Not Found").
3. **Text-Based**: HTTP requests and responses are text-based, making them human-readable. This simplicity makes debugging and development easier.
4. **Stateless Communication**: Since HTTP does not maintain the state, each request is treated as an independent transaction. However, methods like cookies, sessions, and authentication tokens are used to maintain stateful interactions when necessary.
5. **Protocol Versions**: The major versions of HTTP are:

- **HTTP/1.0**: The first version of the protocol. It uses a new TCP connection for each request.
- **HTTP/1.1**: The most widely used version today. It supports persistent connections (i.e., one connection for multiple requests) and allows for more efficient communication.

- **HTTP/2**: Introduces multiplexing, allowing multiple requests and responses to be sent in parallel over a single TCP connection, thus improving performance by reducing latency.
- **HTTP/3**: The latest version that replaces TCP with **QUIC**, a protocol designed for faster and more reliable performance, particularly in the presence of high latency or network congestion.

HTTP Request and Response Structure

1. HTTP Request:

An HTTP request consists of:

- **Request Line**: Contains the HTTP method (e.g., GET, POST), the requested resource (e.g., a URL), and the HTTP version (e.g., HTTP/1.1).
- **Headers**: Metadata about the request such as the browser type (User-Agent), accepted content types (Accept), cookies, authentication tokens, etc.
- **Body**: This is optional and may contain data (e.g., form submissions) in POST or PUT requests.

Example of a GET request:

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
```

2. HTTP Response:

An HTTP response consists of:

- **Status Line**: Contains the HTTP version, a status code (e.g., 200, 404), and a status message (e.g., "OK", "Not Found").
- **Headers**: Metadata about the response, such as the type of content (Content-Type), content length (Content-Length), caching instructions (Cache-Control), etc.
- **Body**: The content of the response, such as the requested HTML page, image, JSON data, or an error message.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 3052
<html>...</html>
```

HTTP Status Codes:

HTTP responses include **status codes** that indicate the result of the request. These codes are grouped into five categories:

- **1xx (Informational)**: Request received, continuing process (e.g., 100 Continue).

- **2xx (Success)**: The action was successfully received, understood, and accepted (e.g., 200 OK, 201 Created).
- **3xx (Redirection)**: Further action needs to be taken to complete the request (e.g., 301 Moved Permanently, 302 Found).
- **4xx (Client Error)**: The request contains bad syntax or cannot be fulfilled (e.g., 404 Not Found, 400 Bad Request).
- **5xx (Server Error)**: The server failed to fulfill a valid request (e.g., 500 Internal Server Error, 503 Service Unavailable).

Significance of HTTP:

1. **Foundation of Web Communication**: HTTP is the backbone of web communication, enabling the transfer of resources across the internet. It defines how web browsers and servers interact, making it essential for web browsing and all internet-based services.
2. **Scalability**: HTTP is a lightweight protocol that can scale easily, allowing millions of devices and users to communicate with each other over the internet without requiring complex state management on the server.
3. **Flexibility**: HTTP supports a wide range of content types (HTML, JSON, XML, images, etc.), making it a flexible protocol for various web-based applications.
4. **Compatibility**: HTTP is supported by virtually all web browsers, servers, and devices, ensuring that websites and services can be accessed globally by users on diverse platforms.
5. **Performance Optimization**: With the introduction of HTTP/2 and HTTP/3, performance improvements like multiplexing and header compression have significantly reduced latency and improved the user experience.

HTTP Stack (Application Layer Protocol Stack)

The **HTTP stack** is the set of technologies and protocols that work together to enable HTTP-based communication on the web. Here's an overview of the different layers of the HTTP protocol stack:

1. **Application Layer**:
   - **HTTP/HTTPS**: At this layer, the actual HTTP requests and responses are created and interpreted. Secure HTTP (HTTPS) is an extension of HTTP that adds encryption using **TLS/SSL**.
   - **DNS (Domain Name System)**: Resolves human-readable domain names (e.g., www.example.com) into IP addresses. DNS helps the browser find the server that hosts the requested web resource.

2. **Transport Layer**:
   - **TCP**: HTTP relies on TCP (Transmission Control Protocol) for reliable data transfer. TCP guarantees that packets are delivered in the correct order and ensures no data is lost.
   - **TLS/SSL**: For HTTPS, encryption is added through the TLS/SSL protocol to secure the communication between the client and server.
3. **Network Layer**:
   - **IP (Internet Protocol)**: This layer handles routing of data packets between different devices on the internet. Each device has a unique IP address, which helps in delivering data to the correct destination.
4. **Data Link Layer**:
   - **Ethernet, Wi-Fi, etc.**: These are the technologies used to physically transmit data over local networks. In wireless or wired networks, data is transferred between devices via protocols like Ethernet or Wi-Fi.
5. **Physical Layer**:
   - **Cables, Radio Waves, Fiber Optics**: This layer defines the physical mediums through which data travels, such as fiber optics for high-speed internet, copper cables, or radio waves for wireless connections.

The **HTTP Protocol** is fundamental to the functioning of the World Wide Web, providing the mechanism by which clients and servers communicate. It is stateless and uses a request-response model. The protocol's stack includes several layers that work together to enable web browsing, from DNS resolution to data transmission over TCP/IP, to encryption over SSL/TLS. With versions like HTTP/2 and HTTP/3, performance and efficiency are continually improving, making it crucial for modern web applications.

HTML Basic Structure

Example:

Key Sections:

1. **<!DOCTYPE html>:** This declaration tells the browser that the document is an HTML5 document.
2. **<html lang="en">:** The root element of the page, with lang="en" indicating the language is English.

3. **<head>:** Contains meta-information about the document, including the title and character encoding. The **<meta>** tag for viewport settings helps make the page responsive on mobile devices.
4. **<body>:** This is where the visible content of the page goes. Everything inside the <body> tag will be displayed in the browser.
5. **<header>:** Typically used for the top part of the page, like a logo or heading.
6. **<nav>:** Defines a navigation menu for the page, which can link to different sections.
7. **<section>:** Divides the page into sections (Home, About, Contact). Each section can have its own content and a unique ID for linking.
8. **<footer>:** Contains footer information like copyright and other legal or contact details.
9. **<a href="mailto:someone@example.com">:** This creates a mailto link, which allows users to click and send an email directly from the page.

Explanation:

1. **HTML Elements:** HTML uses elements to describe the structure of a page. Each element is enclosed in opening and closing tags, e.g., <h1>Welcome</h1>.
2. **Headings:** Use <h1> for the main heading and <h2>, <h3>, etc., for subheadings. Headings help structure the content.
3. **Links:** The <a> tag creates links, and the href attribute specifies the destination of the link.
4. **Lists:** The <ul> tag is for unordered lists, and <li> defines list items. Use it for navigation or other lists.
5. **Paragraphs:** Use the <p> tag for paragraphs of text.

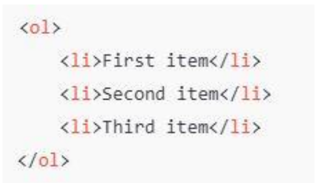
Different types of HTML lists

HTML provides three primary types of lists: **Ordered Lists (ol)**, **Unordered Lists (ul)**, and **Description Lists (dl)**. Each list type has its specific use case and structure. Here's an explanation of each, along with examples:

1. Ordered List (<ol>)

An **Ordered List** is a list where the order of items matters. Items in an ordered list are numbered automatically, and you can specify the starting number and type of numbering (decimal, roman, etc.).

Syntax:



```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
</ol>
```

Example:

Attributes for <ol>:

- type: Defines the numbering style:

  1: Default (Decimal, 1, 2, 3…)

  A: Uppercase letters (A, B, C…)

  a: Lowercase letters (a, b, c…)

  I: Uppercase Roman numerals (I, II, III…)

  i: Lowercase Roman numerals (i, ii, iii…)

Example of a Roman numeral ordered list:

2. Unordered List (<ul>)

An **Unordered List** is a list where the order of items doesn't matter. Items in an unordered list are typically marked with bullet points by default.

Syntax:

```
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>
```

Example:

Attributes for <ul>:

- type: Specifies the style of the bullet points:
  - disc: Default solid circle (•)
  - circle: Hollow circle (○)
  - square: Solid square (▪)

  Example of using a different bullet style:

3. Description List (<dl>)

A **Description List** is used for listing terms and their descriptions. It is a good choice when you want to define a series of items, such as in a glossary or FAQ.

Syntax:

```
<dl>
    <dt>Term 1</dt>
    <dd>Definition of Term 1</dd>
    <dt>Term 2</dt>
    <dd>Definition of Term 2</dd>
</dl>
```

Example:

You can also use **multiple <dd> tags** under a single <dt> if the term has more than one definition or description.

Key Points to Remember:

1. **Ordered Lists (<ol>)** are used when the order of items matters, and they are automatically numbered.
2. **Unordered Lists (<ul>)** are used when the order doesn't matter, and they display items with bullet points.
3. **Description Lists (<dl>)** are used for terms and their definitions or descriptions, typically used in glossaries or FAQs.

Nested Lists in HTML:

1. Nested Unordered List (<ul> inside <ul>)

An **unordered list** can be nested inside another unordered list. Each item in the outer list can have its own nested list.

Example:

2. Nested Ordered List (<ol> inside <ol>)

You can also nest an **ordered list** inside another ordered list. This is useful when the order of items is important, and you want to maintain the numbering for sub-items.

Example:

3. Nested Description List (<dl> inside <dl>)

You can also nest **description lists**. This can be helpful if you want to define terms and their descriptions, and then further break down the descriptions into more detailed explanations.

Example:

Image tag with all its attributes in HTML:

In HTML, the <img> tag is used to embed an image on a webpage. The <img> tag is a self-closing tag and does not have an end tag. It requires several attributes to function properly, with the most essential ones being src, alt, and width/height.

Example-1:

Example-2:

Basic Home Page of a University using HTML concepts:

Example:

Design a university application form for applying to an M.Tech course, collecting various details like Name, Highest Degree, Gender, PGCETRank, Courses applied for (List), Interests (Checkboxes), with a Submit button leading to a Thank You page.

HTML Tables and Frames

HTML tables are used to display tabular data in a structured format. A table is created using the <table> tag, and various other elements help define rows, columns, and the headers of the table. Below is a detailed explanation of HTML tables, along with examples and sample outputs.

1. Basic Structure of a Table

A basic HTML table consists of the following elements:

- <table>: Defines the table.
- <tr>: Defines a row in the table. (table row)
- <th>: Defines a header cell, usually bold and centered. (table header)
- <td>: Defines a standard cell in a row. (table data)

Example:

2. Adding Table Headers

In the example above, the first row <tr> contains <th> elements, which are used to define the table headers. Table headers are typically bold and centered by default.

3. Adding More Complex Data (Nested Tables)

You can also have a table within a table (nested tables). This might be useful for more complex layouts, though it's not commonly used for simple tabular data.

4. Adding Table Captions

You can add a caption to a table for better context. The <caption> tag is used for this purpose, and it appears above the table.

5. Table with Column Span (colspan) and Row Span (rowspan)

The colspan attribute allows a cell to span across multiple columns, while the rowspan attribute allows a cell to span across multiple rows.

Above Table:

Comparison between row span and col span in HTML tables:

Attribute	Rowspan	Colspan
Function	Spans a cell vertically across multiple rows.	Spans a cell horizontally across multiple columns.
Syntax	<td rowspan="n">	<td colspan="n">
Direction	Affects the vertical direction (rows).	Affects the horizontal direction (columns).

Attribute	Rowspan	Colspan
Use Case	Used when one piece of data spans multiple rows.	Used when one piece of data spans multiple columns.
Visual Impact	Creates taller cells by merging rows.	Creates wider cells by merging columns.

In HTML tables, the attributes **rowspan** and **colspan** are used to control how cells span across multiple rows or columns. Both attributes allow you to merge or expand a cell to occupy more than one row or column in the table. Let's look at each attribute in detail, compare them, and provide examples to demonstrate their use.

1. Rowspan

- **Definition**: The rowspan attribute is used in a <td> or <th> element to specify the number of rows a cell should span vertically. This means that the cell will extend vertically across multiple rows, combining several rows into one cell.
- **Use Case**: This is helpful when you want to group data vertically, such as when one piece of data is applicable to several rows, like a shared category for a list of items.

```
<td rowspan="n">Cell content</td>
```

Example:

2. Colspan

- **Definition**: The colspan attribute is used in a <td> or <th> element to specify the number of columns a cell should span horizontally. This means that the cell will extend horizontally across multiple columns, combining several columns into one cell.
- **Use Case**: This is useful when you want to group data horizontally or create headers that span multiple columns, like a section heading that covers several categories.

```
<td colspan="n">Cell content</td>
```

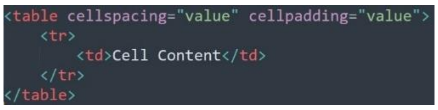
Example:

Both:

Table using Cellspacing and Cellpadding:

cellspacing: Specifies the space between the cells of a table.

cellpadding: Specifies the space between the cell content and the cell border.



```
<table cellspacing="value" cellpadding="value">
    <tr>
        <td>Cell Content</td>
    </tr>
</table>
```

Example:

Frames in HTML

Concept of Frames in HTML:

In HTML, **frames** were traditionally used to display multiple documents or web pages within a single browser window. The idea was to divide the window into different sections, each of which could display content from different sources, like multiple webpages or parts of a single webpage. This was typically achieved using the <frameset> tag, along with the <frame> tag.

However, **frames** have been largely deprecated in modern web design in favor of **CSS-based layouts** and **iframes** for embedding content from external sources. This shift happened because frames had many limitations, such as accessibility issues, poor search engine optimization (SEO), and a lack of flexibility for responsive design.

Still, it's important to understand the concept of frames, as it was a significant part of early web development. Below, I will explain the **frame-based layout** with examples, and then explain the modern alternatives to using frames in HTML.

1. Frames in HTML (Deprecated)

A **frame-based layout** uses a <frameset> tag to divide the browser window into several parts, each containing a different HTML document. Each part is called a **frame**, and each frame can load a separate webpage.

Basic Structure of Frames:

- **<frameset>**: Defines the layout of the frames.
- **<frame>**: Defines a single frame and specifies the document to be displayed in that frame.

Frames Example-1:

Target Frames Example:

iframes Example:

HTML Forms

An HTML form is a fundamental building block of web development that allows users to interact with and submit data to a web application. Forms are commonly used for tasks such as login, registration, and collecting user feedback. An **HTML form** is used to collect user input and submit it to a web server for processing. Forms are commonly used in web applications for activities like logging in, submitting feedback, or placing orders. The form elements allow users to input various types of data, which can then be processed or stored.

Basic Structure of an HTML Form:

A form in HTML is created using the <form> tag, and it can contain various types of input fields. Here's the basic structure:

```
<form action="your-server-endpoint" method="POST">
    <!-- Form elements go here -->
</form>
```

- **action**: Specifies the URL (server endpoint) where the form data will be sent.
- **method**: Defines the HTTP method for form submission. The two most common methods are:

  - **GET**: Sends data as part of the URL (visible). Suitable for non-sensitive data. The GET method appends the form data to the URL as query parameters. This method is commonly used for search forms or when retrieving data from the server. It is visible in the URL and should not be used for sensitive data.
  - **POST**: Sends data in the body of the request (hidden). Suitable for sensitive or large amounts of data. The POST method sends the form data in the body of the HTTP request, which is not visible in the URL. This method is commonly used for submitting sensitive or large amounts of data, such as login forms or any form where data will be processed and stored on the server.

Comparison of GET vs POST

Feature	GET Method	POST Method
Visibility of Data	Data is visible in the URL (query string).	Data is sent in the body of the request (hidden).
Data Length	Limited by the URL length (usually around 2048 characters).	No such length limitation (can be large data).
Security	Not secure, because data is visible in the URL.	More secure, as data is hidden in the request body.
Caching	Can be cached by the browser.	Cannot be cached.
Bookmarks	Can be bookmarked because data is in the URL.	Cannot be bookmarked as the data is not in the URL.
Use Cases	Suitable for non-sensitive data, like search queries, or when retrieving information.	Suitable for sensitive data like login forms or any data that alters the server's state (e.g., creating a record).

GET Method Example:

POST Method Example:

HTML Form Elements

These are the various tags used to create different parts of a form. Some common form elements are:

1. **<form>:** Defines the HTML form itself.
2. **<input>:** Defines an input field where the user can enter data.
3. **<textarea>:** Defines a multi-line text input field.
4. **<select>:** Defines a dropdown list.
5. **<button>:** Defines a clickable button.
6. **<label>:** Defines a label for an input element.
7. **<fieldset>:** Groups related elements in a form.
8. **<legend>:** Defines a caption for the <fieldset> element.
9. **<optgroup>:** Groups options in a dropdown list (<select>).
10. **<datalist>:** Provides predefined options for <input> elements.

Input Type Attributes:

These are specific attributes used with the <input> element to define the type of data the input field will accept. Some common input types are:

1. **text:** A single-line text input field.
2. **password:** A text input field where the characters are masked.
3. **email:** A text field that accepts only email addresses.
4. **number:** A field for numeric input.
5. **date:** A field for date selection (calendar pop-up).
6. **radio:** A radio button that allows selecting one option from a group.
7. **checkbox:** A checkbox input where multiple options can be selected.
8. **file:** Allows file uploads.
9. **button:** A button input used to trigger actions.
10. **submit:** A button to submit the form.
11. **reset:** A button to reset form fields to their default values.
12. **hidden:** A field that is not visible to the user but holds data to be sent with the form.
13. **tel:** A field for telephone number input.

Elements&inputype attrbutes example:

Difference between the Action tag and Onsubmit in the <Form> tag:

The **action** attribute and the **onsubmit** attribute in the <form> tag serve different purposes when it comes to handling form submissions in HTML. Here's a detailed explanation of their differences:

1. action Attribute

The **action** attribute specifies the URL or path where the form data should be sent when the form is submitted. It defines the destination for the form data.

Key Points:

- **Purpose:** The action attribute tells the browser where to send the form data when the user submits the form.
- **Value:** It contains the URL or the path to the server-side script (e.g., a PHP, Python, or Node.js script) or a web address that will process the submitted form data.
- **Default Behavior:** If the action attribute is omitted, the form data will be sent to the same URL that the form was submitted from.

Example:

2. onsubmit Attribute

The **onsubmit** attribute specifies a JavaScript function or code to run when the form is submitted. It allows you to execute custom actions before or after the form submission, such as validation, logging, or modifying form data.

Key Points:

- **Purpose:** The onsubmit attribute allows you to specify JavaScript code that will be executed when the form is submitted. This is usually used to perform client-side form validation or to handle submission asynchronously (AJAX).
- **Value:** The value of the onsubmit attribute is a JavaScript function or a script that runs when the form is submitted.
- **Default Behavior:** If the JavaScript in onsubmit returns false, it will prevent the form from submitting. This is useful for validation purposes.

Example-1:

Example-2:

Key Differences Between action and onsubmit:

Feature	action Attribute	onsubmit Attribute
Purpose	Specifies where to send the form data when it is submitted.	Defines JavaScript code that runs when the form is submitted.
Location	Always inside the <form> tag, typically as a URL.	Also inside the <form> tag, specifying a JavaScript function or script.
Usage	Used for defining the endpoint to process form data.	Used for validating, modifying, or controlling the form submission process using JavaScript.
Default Action	If omitted, submits to the current page URL.	Runs the JavaScript code on form submission.
Behavior Control	Does not control form submission directly.	Can prevent or alter the form submission by returning false or true.
Common Use	Directs form data to the server (URL, script).	Used for client-side validation or handling form submission with JavaScript (AJAX).

Model multiple forms on a single page:

Difference between anchor buttons and image buttons:

1. Anchor Buttons (Without CSS)

An **anchor button** is simply a link (<a>) that is styled or used to look like a button. When clicked, it typically navigates to another page or performs a hyperlink action.

2. Image Buttons (Without CSS)

An **image button** uses an image to act as a clickable button. You typically use this when you want a graphical button (such as an icon or a custom design) to trigger an action, such as submitting a form.

Key Differences Between Anchor Buttons and Image Buttons (Without CSS)

Attribute	Anchor Button	Image Button
HTML Tag	<a> (anchor)	<input type="image">
Primary Function	Navigation (hyperlink to another page or location)	Action (usually form submission or triggering actions)
Action on Click	Opens a hyperlink or executes a JavaScript function	Submits a form or triggers a specific action
Usage Context	Hyperlinks, navigation, anchor links	Form submission, graphical buttons (e.g., icons)

Difference between a Button and a Submit Button.

1. Button (<button>)

The <button> element is a generic button in HTML that can be used for a variety of purposes. It can be used to trigger actions like running JavaScript functions, submitting forms (when specifically set to type="submit"), or triggering other events.

Key Characteristics of a Button:

- **Purpose**: A generic button that can trigger any action, including running JavaScript or submitting a form.
- **HTML Tag**: <button>
- **Action**: The action depends on the type attribute. By default, it's a **generic button** that requires JavaScript or other events to perform actions.
- **Use Cases**: Can be used for custom actions like executing JavaScript, opening modals, or changing UI elements dynamically.

2. Submit Button (<input type="submit">)

A **submit button** is specifically designed for submitting forms. It triggers the submission of the form to the server for processing.

Key Characteristics of a Submit Button:

- **Purpose**: Specifically designed to submit forms to the server for processing.
- **HTML Tag**: <input type="submit">
- **Action**: Submits the form data to the URL specified in the action attribute of the form.
- **Use Cases**: Used only to submit data in forms. When clicked, it triggers form submission.

HTML Elements and Attributes

Illustration the syntax for Radio Button Groups and Checkboxes.

Radio Button Groups:

A radio button group allows users to select one option from a list of mutually exclusive choices. Only one radio button in a group can be selected at a time.

Syntax:

Example:

Checkboxes:

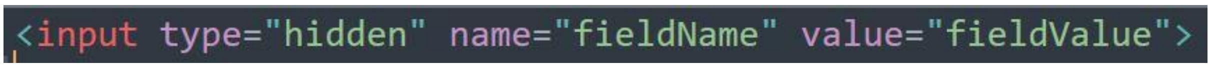
A checkbox allows users to select one or more options from a list. Multiple checkboxes can be selected simultaneously.

Syntax:

Example:

Use of the Hidden Field in HTML:

The hidden field in HTML is created using the <input> tag with the attribute type="hidden". It is used to store data that is not visible to the user but is sent to the server when the form is submitted.



```
<input type="hidden" name="fieldName" value="fieldValue">
```

type="hidden": Specifies that the input is a hidden field.

name: Assigns a name to the field, used as the key when the data is submitted.

value: Holds the value to be sent with the form submission.

Example:

Use of the <SUB> and <SUP> tags:

The <sub> and <sup> tags in HTML are used to display text as subscript and superscript, respectively. These are commonly used in mathematical formulas, chemical equations, and footnotes.

Subscript (<sub>):

<sub>Subscript Text</sub>

Superscript (<sup>):

<sup>Superscript Text</sup>

Example:

Use of the <PRE> tag:

The <pre> tag in HTML is used to display preformatted text. It preserves the formatting of the content, including spaces, line breaks, and tabs, exactly as it appears in the source code. This is especially useful for displaying code snippets, ASCII art, or any text where spacing is crucial.

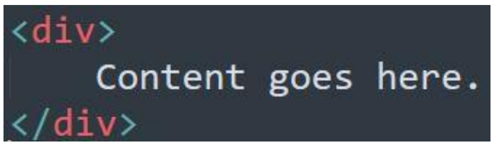
Syntax:

Example:

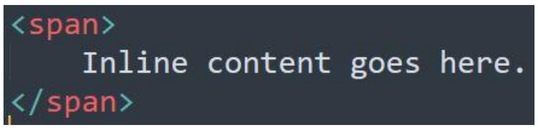
Key Differences Between <div> and <span>

Aspect	<div>	<span>
Purpose	Used as a **block-level** container for grouping larger sections of content.	Used as an **inline-level** container for grouping small pieces of text or inline elements.
Default Display	Block-level: starts on a new line and takes up the full width available.	Inline-level: remains on the same line as surrounding content.
Use Case	For layout and structural purposes, often used with CSS for dividing content into sections.	For styling or modifying small portions of text or inline elements.

Aspect	<div>	<span>
Styling	Used with CSS for larger areas of the page (e.g., sections, containers).	Used for styling or highlighting smaller elements (e.g., words, phrases).



```
<div>
    Content goes here.
</div>
```



```
<span>
    Inline content goes here.
</span>
```

<div> tag example:

<span> tag example:

HTTP and URLs

URLs:

A URL (Uniform Resource Locator) is the address used to access resources on the internet. It acts as the "location" of a web page, image, video, file, or any other resource hosted online. URLs are a core part of the World Wide Web, enabling users to navigate and interact with content.

Structure of a URL

A URL typically has the following components:

scheme://username:password@hostname:port/path?query#fragment

Component	Description	Example
Scheme	Specifies the protocol used to access the resource (e.g., http, https, ftp).	https
Username:Password (optional)	Credentials for accessing resources that require authentication. (Rarely used in modern URLs.)	username:password@
Hostname	The domain name or IP address of the server hosting the resource.	www.example.com
Port (optional)	Specifies the port number on the server (default ports: 80 for HTTP, 443 for HTTPS).	:8080
Path	The specific location of the resource on the server.	/products/page.html
Query String (optional)	A string of key-value pairs to pass additional information to the server.	?id=123&sort=asc
Fragment (optional)	A reference to a specific section of a page (anchors, usually).	#section1

Example of a URL

Consider the following URL:

https://www.example.com:8080/products/page.html?id=123&sort=asc#details

Scheme: https

Specifies that the resource is accessed securely via HTTPS.

Hostname: www.example.com

The server's domain name.

Port: 8080

A specific port on the server (default ports like 80 or 443 are often omitted).

Path: /products/page.html

Indicates the location of the resource on the server.

Query String: ?id=123&sort=asc

Parameters sent to the server (id=123 and sort=asc).

Fragment: #details

Points to a specific section (e.g., a <div> with id="details" in the page).

Types of URLs

1. **Absolute URL**

A complete URL that contains all necessary information to locate a resource.

Example:

https://www.example.com/index.html

2. **Relative URL**

Specifies a path relative to the current page or directory.

Example:

If the current URL is https://www.example.com/products/, a relative URL like page.html would point to https://www.example.com/products/page.html.

HTTP vs. HTTPS:

HTTP (HyperText Transfer Protocol):

The standard protocol for transferring data over the web.

HTTPS (HTTP Secure):

A secure version of HTTP that uses encryption (via SSL/TLS) to protect data during transmission.

Uses of URLs:

- ✓ Access Resources: Locate web pages, files, images, videos, or APIs.

- ✓ Query Parameters: Pass information between a client and a server (e.g., search terms, user IDs).
- ✓ Bookmarking: Save and share specific pages or sections.
- ✓ API Endpoints: Interact with web services by sending requests to defined URLs.

Common URL Issues:

- ✓ Broken Links: URLs that no longer point to a valid resource.
- ✓ Case Sensitivity: Some servers are case-sensitive for URLs (e.g., /File.html vs. /file.html).
- ✓ Special Characters: Characters like spaces and # must be encoded (%20 for space).
- ✓ Redirection: A URL may redirect users to another location (e.g., using HTTP 301 or 302 codes).

Importance of URLs:

- ✓ Navigation: Allow users to locate and access resources on the web.
- ✓ SEO (Search Engine Optimization): Well-structured URLs help search engines understand and rank pages.
- ✓ Security: HTTPS ensures data privacy and trustworthiness.
- ✓ Customization: Short and meaningful URLs enhance user experience and branding.

Request and Response message in HTTP:

1. HTTP Request Message

An HTTP Request is sent by the client to the server to request a resource or perform an action. It typically consists of the following components:

Components of an HTTP Request:

a. Request Line

This line includes the HTTP method, the requested URL, and the HTTP version.

- ✓ **HTTP Method:** Specifies the action to be performed. Common methods include:
- ✓ **GET:** Retrieve a resource.
- ✓ **POST:** Submit data to the server (e.g., form submissions).
- ✓ **PUT:** Replace a resource.
- ✓ **DELETE:** Remove a resource.
- ✓ **PATCH:** Update a resource partially.
- ✓ **Request URL:** The URL of the resource being requested. This could be a path, domain, or endpoint on the server (e.g., /home or https://www.example.com/resource).
- ✓ **HTTP Version:** Specifies the version of the HTTP protocol being used (e.g., HTTP/1.1 or HTTP/2).

b. Headers

HTTP headers provide additional metadata about the request, such as the type of content being sent, supported languages, or the user-agent (browser) making the request.

Common headers:

- ✓ **Host:** Specifies the domain name of the server.
- ✓ **User-Agent:** Identifies the client software (browser) making the request.
- ✓ **Content-Type:** Indicates the media type of the request body (for POST or PUT requests).
- ✓ **Accept:** Lists the content types the client can process (e.g., text/html, application/json).
- ✓ **Authorization:** Used for passing credentials (e.g., API tokens or Basic Auth).
- ✓ **Accept-Encoding:** Lists the content encodings the client can accept (e.g., gzip, deflate).

c. Body (Optional)

The body of the HTTP request contains the data sent to the server. This is typically included in POST, PUT, or PATCH requests to send data such as form submissions, file uploads, or JSON data.

For example, in a form submission, the body would include the form fields and values.

Example of an HTTP Request Message:

GET /products HTTP/1.1

Host: www.example.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36

Accept: application/json

2. HTTP Response Message

An HTTP Response is sent by the server to the client in response to an HTTP request. It contains the status of the request (whether it was successful or not) and, if applicable, the requested resource or an error message.

Components of an HTTP Response:

a. Status Line

The status line consists of the HTTP version, a status code, and a status message. The status code indicates the result of the request.

- ✓ HTTP Version: Specifies the version of the HTTP protocol being used (e.g., HTTP/1.1 or HTTP/2).
- ✓ Status Code: A 3-digit code indicating the result of the request (e.g., 200 OK, 404 Not Found).

- ✓ Status Message: A short phrase providing a human-readable explanation of the status code.

b. Headers

The response headers provide metadata about the response, such as the type of content returned, caching directives, and server details.

Common headers:

- ✓ Content-Type: Specifies the type of content being returned (e.g., text/html, application/json).
- ✓ Content-Length: The size of the response body in bytes.
- ✓ Cache-Control: Directives for caching mechanisms.
- ✓ Set-Cookie: Contains cookies to be set on the client-side.
- ✓ Location: Used for redirection responses (e.g., 301 Moved Permanently).

c. Body

The body of the response contains the data that was requested or the result of the action taken. This could include HTML content, JSON data, an image file, or a simple error message, depending on the request.

For example, a GET request may return the HTML content of a webpage, while a POST request may return a confirmation message or an updated resource.

Example of an HTTP Response Message:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 123
Cache-Control: no-cache
{
  "status": "success",
  "message": "Request processed successfully"
}
```

Request-Response Cycle Example:

1. **Client Sends Request:** A client (e.g., a web browser) sends an HTTP request to a server for a specific resource, such as a webpage.
2. **Server Processes Request:** The server processes the request. If the resource exists, the server retrieves it from storage. If not, it generates an error (e.g., 404 Not Found).

3. **Server Sends Response:** The server responds with a status code indicating whether the request was successful and returns the requested resource or an error message.

HTTP request and response messages form the core of web communication. The request is made by the client, containing the method, headers, and optional body, while the server responds with the status code, headers, and an optional body containing the requested resource or an error message. Proper formatting and handling of these messages ensure seamless communication between clients and servers on the web.

HTML5 Features

HTML5 introduced many new features, elements, and APIs designed to improve the structure, functionality, and usability of web pages. Below is an illustration of some of the key new features in HTML5 with complete examples and explanations.

1. **Semantic Elements**
2. **Form Enhancements**
3. **Audio and Video Support**
4. **Canvas Elements**
5. **Local Storage**
6. **Geolocation API**
7. **Web Workers**
8. **Responsive Images**
9. **Drag and Drop API**
10. **Web Sockets**
11. **Micro Data**
12. **Cross Document Messaging**

1. **New Semantic Elements**

HTML5 introduced several new semantic elements to help structure web content more meaningfully and improve accessibility and SEO.

Explanation:

- ☑ **<header>:** The <header> element represents introductory content or a set of navigational links. It is typically used to group the introductory content of a page or a section (such as a header section with logos, navigation, and introductory text).
- ☑ **<nav>:** The <nav> element represents a navigation section containing links to other pages or sections within the same document. It is used to group together links that help users navigate the website.

- ☑ **<article>:** The <article> element represents a self-contained piece of content that can be independently distributed or reused (e.g., in syndication). This could be a blog post, news article, forum post, etc.
- ☑ **<aside>:** The <aside> element represents content that is tangentially related to the content around it. This is often used for sidebars, callouts, or information that complements the main content but is not essential to it.
- ☑ **<footer>:** The <footer> element represents the footer section of a document or section. It typically contains information about the author, copyright, contact links, or related documents.
- ☑ **<section>:** The <section> element is used to represent a section of content in a document, typically a thematic grouping of content. It can be used to divide the page into distinct sections, each with its own heading

These elements help to organize content logically and are beneficial for SEO and screen readers.

2. Form Enhancements

HTML5 introduces new input types that offer better user experience and form validation.

Explanation:

- ☑ **type="email":** Ensures the input is a valid email address.
- ☑ **type="date":** Provides a date picker for users.
- ☑ **type="url":** Ensures the input is a valid URL.
- ☑ **type="number":** Allows only numerical input with validation for a range (min and max).

These input types offer better user interaction and validation without needing additional JavaScript.

3. Audio and Video Elements

HTML5 introduced native support for embedding audio and video content without the need for third-party plugins like Flash.

Explanation:

- ☑ **<audio>:** Allows embedding audio content. The controls attribute provides play, pause, and volume controls.
- ☑ **<video>:** Allows embedding video content. It also supports controls like play, pause, and volume control.

These elements offer native support for multimedia content.

4. Canvas Element

The <canvas> element allows dynamic, scriptable rendering of 2D shapes and bitmap images.

Explanation:

☑ **<canvas>:** Defines a drawing surface. The getContext("2d") method provides a 2D drawing context.

The script demonstrates drawing a red rectangle, blue circle, and text on the canvas.

This allows for creating interactive graphics, games, and animations directly in the browser.

5. Local Storage and Session Storage

HTML5 provides localStorage and sessionStorage APIs to store data on the client side.

Explanation:

☑ **localStorage.setItem(key, value):** Saves data with a specific key.

☑ **localStorage.getItem(key):** Retrieves data by key.

This data persists even after the user closes the browser.

6. Geolocation API

HTML5 includes a Geolocation API that allows web applications to access the user's geographical position.

Explanation:

☑ **navigator.geolocation.getCurrentPosition():** Requests the user's current location.

The location is then displayed in the paragraph with the id="location".

This API enables location-based services, such as maps and location tracking.

7. Web Workers

Web Workers allow running scripts in the background, off the main thread, improving performance by preventing UI blocking.

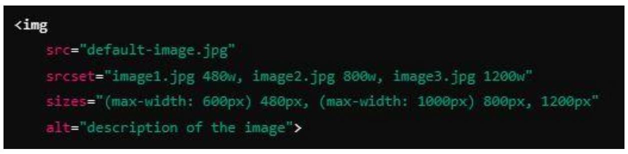
Explanation:

Web Worker: Runs JavaScript in the background thread. The example starts and stops a worker, which sends a message back to the main thread.

This allows running resource-intensive tasks without blocking the UI.

8. Responsive Images

Responsive images adjust automatically to different screen sizes and resolutions. This is essential for mobile-first websites. You can use the srcset and sizes attributes for images to create responsive layouts.



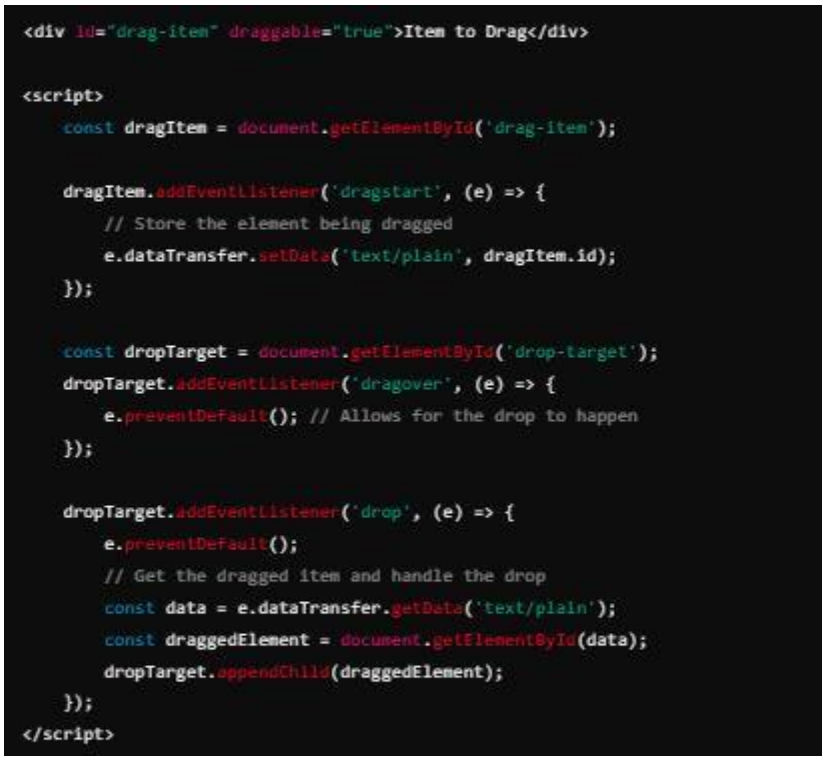
```
<img
    src="default-image.jpg"
    srcset="image1.jpg 480w, image2.jpg 800w, image3.jpg 1200w"
    sizes="(max-width: 600px) 480px, (max-width: 1000px) 800px, 1200px"
    alt="description of the image">
```

Explanation:

- src: Defines the default image to be used if srcset is not supported.
- srcset: Specifies multiple images for different screen widths (e.g., 480w, 800w, etc.).
- sizes: Specifies the image size to use based on viewport width. For example, (max-width: 600px) 480px means if the screen width is 600px or smaller, the image width will be 480px.

9. Drag and Drop API

The Drag and Drop API allows users to drag and drop elements within a web page. It's useful for file uploads, reordering lists, or moving images around.



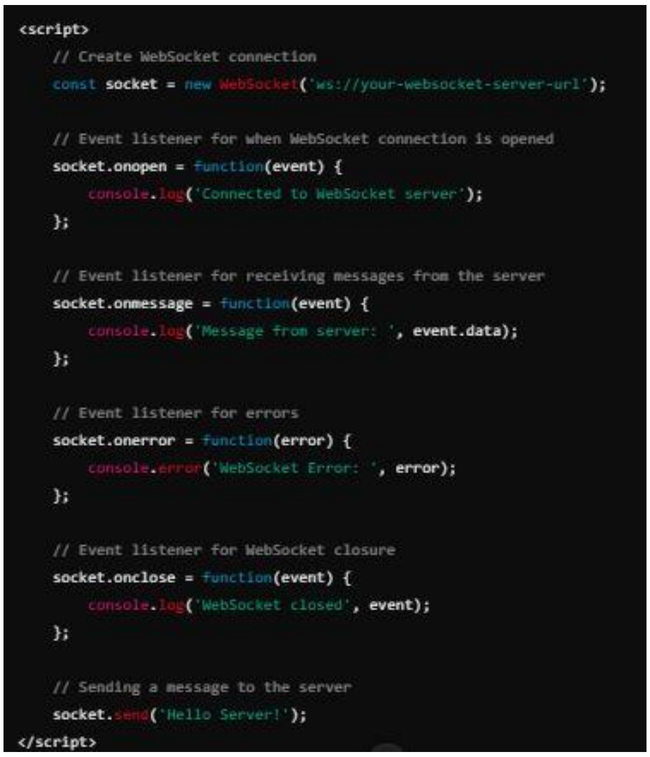
```
<div id="drag-item" draggable="true">Item to Drag</div>

<script>
    const dragItem = document.getElementById('drag-item');

    dragItem.addEventListener('dragstart', (e) => {
        // Store the element being dragged
        e.dataTransfer.setData('text/plain', dragItem.id);
    });

    const dropTarget = document.getElementById('drop-target');
    dropTarget.addEventListener('dragover', (e) => {
        e.preventDefault(); // Allows for the drop to happen
    });

    dropTarget.addEventListener('drop', (e) => {
        e.preventDefault();
        // Get the dragged item and handle the drop
        const data = e.dataTransfer.getData('text/plain');
        const draggedElement = document.getElementById(data);
        dropTarget.appendChild(draggedElement);
    });
</script>
```

Explanation:

- **draggable="true":** Marks the element as draggable.
- **dragstart:** Event triggered when dragging starts.
- **dataTransfer.setData():** Used to store data when dragging.
- **dragover:** Event triggered when the dragged item is over a valid drop target. **e.preventDefault()** is required to enable the drop.
- **drop:** Event triggered when the dragged item is dropped.

10. Web Sockets

Web Sockets allow for bidirectional communication between the client and server in real-time. This is useful for chat applications, live notifications, or online gaming.



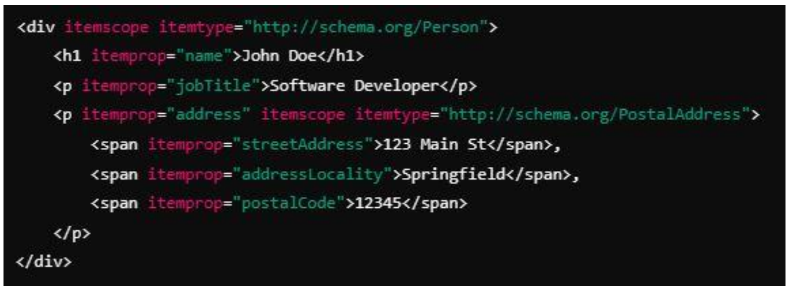
```
<script>
    // Create WebSocket connection
    const socket = new WebSocket('ws://your-websocket-server-url');

    // Event listener for when WebSocket connection is opened
    socket.onopen = function(event) {
        console.log('Connected to WebSocket server');
    };

    // Event listener for receiving messages from the server
    socket.onmessage = function(event) {
        console.log('Message from server: ', event.data);
    };

    // Event listener for errors
    socket.onerror = function(error) {
        console.error('WebSocket Error: ', error);
    };

    // Event listener for WebSocket closure
    socket.onclose = function(event) {
        console.log('WebSocket closed', event);
    };

    // Sending a message to the server
    socket.send('Hello Server!');
</script>
```

Explanation:

- **new WebSocket('ws://your-url'):** Initializes a WebSocket connection.
- **onopen:** Event triggered when the WebSocket connection is established.
- **onmessage:** Event triggered when a message is received from the server.
- **send():** Sends a message to the server.
- **onerror and onclose:** Handle errors and closure events.

11. Micro Data

Microdata is a way of embedding structured data in HTML to make your website more understandable to search engines and other machines. It is used to create machine-readable information in the form of schema.org markup.



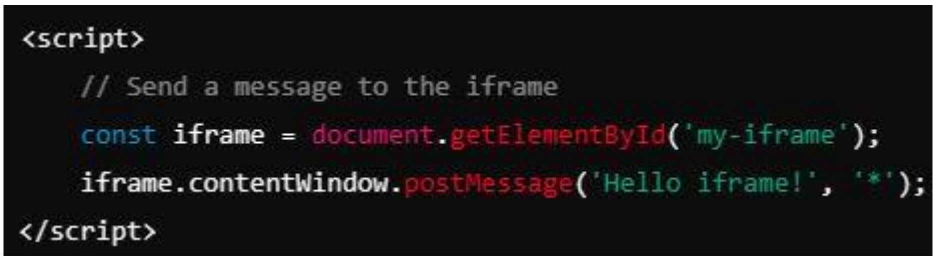
```
<div itemscope itemtype="http://schema.org/Person">
    <h1 itemprop="name">John Doe</h1>
    <p itemprop="jobTitle">Software Developer</p>
    <p itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
        <span itemprop="streetAddress">123 Main St</span>,
        <span itemprop="addressLocality">Springfield</span>,
        <span itemprop="postalCode">12345</span>
    </p>
</div>
```

Explanation:

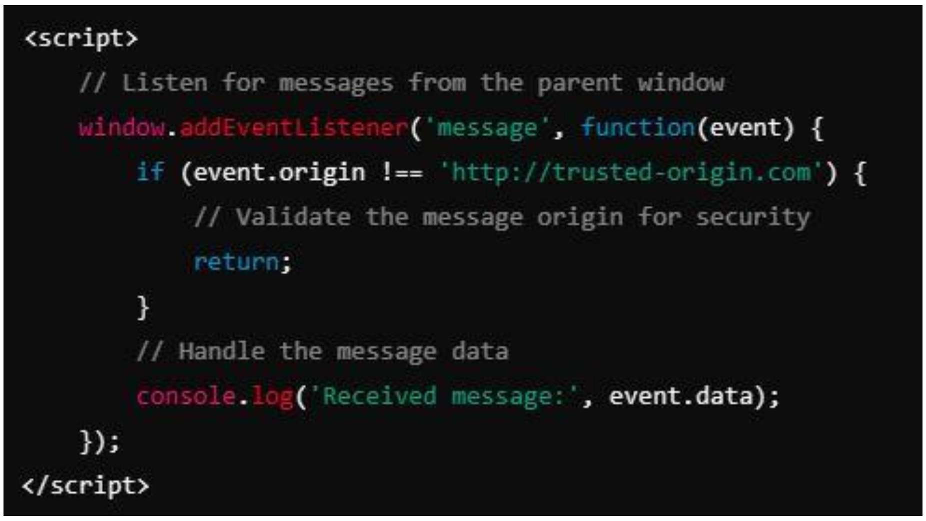
- itemscope: Defines the scope of the data.
- itemtype: Specifies the type of data, using the URL from schema.org.
- itemprop: Associates specific properties of the schema type (e.g., name, jobTitle, address).

12. Cross Document Messaging

Cross-document messaging allows different web pages to communicate with each other, especially between iframes or windows from different origins, using the postMessage() API.



```
<script>
    // Send a message to the iframe
    const iframe = document.getElementById('my-iframe');
    iframe.contentWindow.postMessage('Hello iframe!', '*');
</script>
```



```
<script>
    // Listen for messages from the parent window
    window.addEventListener('message', function(event) {
        if (event.origin !== 'http://trusted-origin.com') {
            // Validate the message origin for security
            return;
        }
        // Handle the message data
        console.log('Received message:', event.data);
    });
</script>
```

Explanation:

- **postMessage():** Used in the parent window to send a message to the iframe.
- **message:** Event listener to handle messages in the iframe.
- **event.origin:** You should always verify the origin of the message to avoid security vulnerabilities.

CSS Basics

CSS:

CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation of a document written in HTML (or XML). It defines how HTML elements should appear on screen, paper, or other media types. CSS helps you control the layout, colors, fonts, spacing, and other visual aspects of a webpage.

CSS separates content from design, meaning you can create the structure and content of a webpage using HTML and use CSS to define its appearance, allowing for easier maintenance and better flexibility.

Key Concepts in CSS:

☑ **Selectors:** These define which HTML elements you want to style.

- Universal Selector (*)
- Element Selector (Type Selector)
- ID Selector (#id)
- Class Selector (.class)
- Attribute Selector
- Descendant Selector (Space between selectors)
- Child Selector (>)
- Adjacent Sibling Selector (+)
- General Sibling Selector (~)
- Pseudo-classes (:pseudo-class)
- Pseudo-elements (::pseudo-element)

Example:

☑ **Properties:** These define the specific aspect of the element you want to change (like color, size, font, etc.).

☑ **Values:** These are the settings for the properties (like a specific color, size, etc.).

Types of CSS:

The "Cascading" Effect:

The "cascading" in CSS comes from the way styles are applied. When multiple styles apply to the same element, CSS decides which one to use based on specificity, importance, and order.

For example:

☑ Inline styles override internal styles.

☑ Internal styles override external styles, unless external styles are marked as !important.

Benefits of Using CSS:

1. **Separation of Concerns:** It separates the structure (HTML) from the design (CSS), making it easier to maintain and update.
2. **Consistency:** You can apply the same styles across multiple pages of a website.
3. **Flexibility:** CSS allows for responsive design, meaning your webpage can look good on any device (desktop, tablet, mobile).
4. **Customization:** CSS provides a lot of flexibility in styling, from layout to animation.

Advantages and disadvantages of CSS with relevant examples:

Advantages:

☑ **Separation of Concerns:**

- CSS separates the structure (HTML) from the design (CSS), making it easier to maintain and update styles.
- Example: A website with multiple pages can use one CSS file, so changes need only to be made in that single file rather than every page.

☑ **Consistency:**

- Styles can be applied consistently across multiple pages.
- Example: All headers can have the same color and font across a website by defining styles in a single CSS file.

☑ **Faster Webpages:**

- Using external CSS files reduces code duplication and file size, leading to faster load times.

☑ **Responsive Design:**

- CSS enables web pages to adapt to different screen sizes, making them responsive.
- Example: The @media query allows different styles for mobile and desktop devices.

Disadvantages:

☑ **Complexity in Large Websites:**

- Managing large amounts of CSS can be cumbersome, especially for big websites.
- Example: Complex stylesheets can become hard to manage if not structured well.

☑ **Browser Compatibility:**

- Different browsers may render the same CSS differently.
- Example: CSS flexbox might work in one browser, but not in another, or older browsers might not support newer CSS features.

Example:

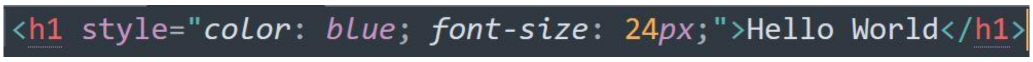
Types of CSS:

CSS can be added to a webpage in three ways:

1. **Inline CSS:**

   CSS is applied directly inside HTML elements using the style attribute.

   Syntax:



```
<h1 style="color: blue; font-size: 24px;">Hello World</h1>
```

   Features:

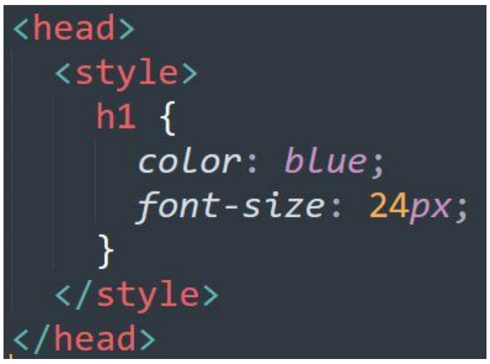
   - Styles are applied directly to individual elements.
   - Easy to implement but not reusable or maintainable.

   Example:

2. **Internal CSS:**

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

   Syntax:



```
<head>
  <style>
    h1 {
      color: blue;
      font-size: 24px;
    }
  </style>
</head>
```

   Example:

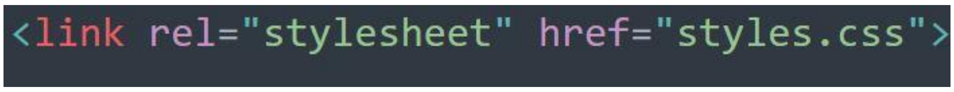
   Features:

   - Styles are written once and applied across multiple elements.
   - Ideal for small projects or single-page applications.

3. **External CSS:**

   CSS is written in an external file, and this file is linked to the HTML document.

   Syntax:



```
<link rel="stylesheet" href="styles.css">
```

   Features:

   - Reusable styles can be applied across multiple pages by linking the same CSS file.
   - Ideal for large projects or websites with consistent designs.

   Example:

Comparison of CSS Types

Feature	Inline CSS	Internal CSS	External CSS
Where Defined	Inside the HTML element	Inside <style> in <head>	In a separate .css file
Usage	Quick, one-time styling	Single-page styles	Multi-page, reusable styles
Maintainability	Hard to maintain	Easier to maintain	Best for maintainability
Reusability	Not reusable	Limited reusability	Highly reusable
Performance	Slower (inline styles inline in DOM)	Moderate	Fast (loads once for all pages)

Best Practice:

☑ Use External CSS for projects with multiple pages to ensure reusability and maintainability.

☑ Use Internal CSS for single-page projects or specific page overrides.

☑ Avoid Inline CSS except for small, one-time changes or for testing purposes.

Use of ID selector in CSS:

In CSS, both id and class are used as selectors to apply styles to HTML elements, but they have different purposes and behaviors.

1. **id in CSS**

☑ **Purpose:** The id attribute is used to select a single, unique element in the HTML document. An id should only be used once per page because it represents a unique identifier for an element.

☑ **Selector:** In CSS, id selectors are prefixed with the **(#)** symbol.

☑ **Specificity:** The id selector has a higher specificity than the class selector. This means if both id and class selectors target the same element, the id styles will override the class styles

Example:

2. **class in CSS**

☑ **Purpose:** The class attribute is used to select one or more elements that share the same class. Unlike id, a class can be used on multiple elements across the page.

☑ **Selector:** In CSS, class selectors are prefixed with the **(.)** symbol.

☑ **Specificity:** The class selector has lower specificity compared to the id selector. This means if both class and id styles target the same element, the id styles will take precedence.

Example:

Combined Example (Using both id and class)

You can use both id and class selectors on the same element, but remember that the id has higher specificity than the class. This combined example demonstrates how to use both id and class together.

Example:

Key Differences Between id and class

Feature	id	class
Purpose	Unique identifier for a single element.	Used for grouping multiple elements.
Uniqueness	Must be unique within a page (only one element can have the same id).	Can be reused on multiple elements.
Usage	Used when you want to apply styles to one specific element.	Used when you want to apply styles to multiple elements.
Specificity	Higher specificity (overrides class selectors).	Lower specificity (can be overridden by id).
Syntax	Prefixed with # (e.g., #my-id).	Prefixed with . (e.g., .my-class).

When to Use id vs class

Use id when:

- ☑ You need to style a unique element.
- ☑ The element appears only once on the page.
- ☑ You need to target an element for JavaScript manipulation (e.g., document.getElementById()).

Use class when:

- ☑ You want to style multiple elements with the same style.
- ☑ The element may appear multiple times on the page.
- ☑ You want to create reusable styles.

CSS properties related to text with a suitable example:

CSS provides several properties to style text, including color, size, alignment, spacing, and decoration. Below are the key CSS properties related to text, explained in detail with examples and comments.

Key CSS Text Properties

- ☑ **color:** Sets the color of the text.
- ☑ **font-family:** Specifies the font type.

- ☑ **font-size:** Sets the size of the text.
- ☑ **font-style:** Specifies the style of the font (normal, italic, or oblique).
- ☑ **font-weight**: Controls the thickness of the font (normal, bold, lighter, or numerical values).
- ☑ **text-align:** Aligns the text horizontally (left, right, center, justify).
- ☑ **text-decoration:** Adds decoration to text (underline, overline, line-through, none).
- ☑ **line-height:** Sets the height of a line of text.
- ☑ **letter-spacing:** Adjusts the space between letters.
- ☑ **word-spacing:** Adjusts the space between words.
- ☑ **text-transform:** Controls the capitalization (uppercase, lowercase, capitalize).
- ☑ **white-space:** Controls how whitespace inside an element is handled (normal, nowrap, pre, etc.).
- ☑ **direction:** Specifies the direction of text (ltr for left-to-right or rtl for right-to-left).

Example:

Explanation:

- ☑ **Basic Text Styling (.basic):**

  Demonstrates color, font-size, font-family, and line-height.

  Sets a serif font for contrast and increases line height for readability.
- ☑ **Text Alignment (.alignment):**

  Uses text-align: justify to align text to both the left and right edges, creating a clean block of text.
- ☑ **Letter and Word Spacing (.spacing):**

  Increases space between letters (letter-spacing) and words (word-spacing) to improve readability or achieve a specific design.
- ☑ **Text Transform and Decoration (.transform-decorate):**

  Uses text-transform: capitalize to make the first letter of each word uppercase.

  Adds an underline using text-decoration.
- ☑ **White-space Control (.whitespace):**

  Demonstrates how white-space: nowrap prevents collapsing of multiple spaces, preserving the original formatting.
- ☑ **Text Direction (.direction and .direction-rtl):**

  The direction: ltr (default) styles left-to-right text.

The direction: rtl applies a right-to-left text direction, useful for languages like Arabic or Hebrew, and aligns text to the right.

CSS properties related to tables with relevant examples:

CSS provides several properties to style tables, including borders, spacing, alignment, and background. These properties can be applied to the table, rows, cells, and headers to control the layout and design.

Key CSS Table Properties:

- ☑ **border:** Sets the border for the table, rows, or cells.
- ☑ **border-collapse:** Combines adjacent cell borders into a single border.
- ☑ **border-spacing:** Specifies the space between borders of adjacent cells.
- ☑ **padding:** Adds space inside table cells.
- ☑ **text-align:** Aligns text inside cells horizontally.
- ☑ **vertical-align:** Aligns text inside cells vertically.
- ☑ **width:** Specifies the width of the table or its cells.
- ☑ **background-color:** Sets the background color of the table, rows, or cells.
- ☑ **font-family and font-size:** Control the font styles within the table.
- ☑ **hover effects:** Adds styles when the user hovers over a row or cell.

Example:

Class Properties in CSS:

In the context of CSS, class properties refer to the various styling attributes (like font size, color, margins, etc.) that you can apply to elements that share the same class. When you define a class in CSS, you can assign it several properties that will dictate the appearance or behavior of any HTML element assigned to that class.

Usage of Class Properties in CSS:

Class properties are used to:

- ☑ Group elements that should share similar styles.
- ☑ Maintain consistency across elements of the same class.
- ☑ Make your code reusable and more maintainable by applying the same style to multiple elements without repeating code.
- ☑ Separate content from styling, making HTML cleaner and easier to read.

Advantages of Using Class Properties in CSS:

- ☑ **Reusability:** You can apply the same class to multiple elements, so you don't have to redefine styles for each individual element.

☑ **Maintainability:** If you want to update the style of all elements with a particular class, you only need to change the CSS in one place.

☑ **Consistency:** It ensures that all elements with the same class look consistent, especially when multiple elements share similar behavior or structure.

☑ **Separation of Concerns:** By using classes in CSS, you separate the content (HTML) from the presentation (CSS), improving readability and making it easier to modify the appearance of the page.

Example:

CSS properties related to the <a> tag and backgrounds:

The <a> tag in HTML represents a hyperlink, which can link to another page, a section within the same page, or an external website. CSS provides various properties to style anchor tags (<a>) and their background properties, allowing web developers to control their appearance when users interact with them.

Below is a detailed explanation of the relevant CSS properties related to the <a> tag and backgrounds, along with an example to demonstrate them:

Example(<a>):

CSS Properties for <a> Tag:

1. **color**

   Defines the color of the text inside the anchor tag.

   Example: To change the color of the link text.

```css
a {
    color: #007bff; /* Blue color */
}
```

2. **text-decoration**

   Controls the text decoration applied to the link. The most common values are none, underline, overline, and line-through.

   Example: Remove the underline from the link.

```css
a {
    text-decoration: none; /* Removes underline */
}
```

3. **font-weight**

Sets the weight of the font for the anchor text. It can take values like normal, bold, lighter, or a numeric value.

Example: Make the anchor text bold.

```
css

a {
    font-weight: bold; /* Makes text bold */
}
```

4. **font-style**

Specifies the font style of the anchor text. Common values are normal, italic, and oblique.

Example: Make the anchor text italic.

```
css

a {
    font-style: italic; /* Makes text italic */
}
```

5. **background-color**

Defines the background color of the anchor tag.

Example: Add a background color to the anchor text.

```
css

a {
    background-color: #f0f0f0; /* Light gray background */
}
```

6. **border**

Adds a border around the anchor tag. This can be customized by setting properties like border-width, border-style, and border-color.

Example: Add a border around the link.

```
css

a {
    border: 1px solid #007bff; /* Blue border around the link */
    padding: 5px; /* Optional: Adds space around the text */
}
```

7. **padding**

Controls the space between the text and the border (if any) of the anchor tag.

Example: Add padding around the link text.

```css
a {
    padding: 10px 20px; /* 10px top/bottom, 20px left/right */
}
```

8. **text-transform**

Alters the case of the anchor text, with values like uppercase, lowercase, and capitalize.

Example: Convert the text of the link to uppercase.

```css
a {
    text-transform: uppercase; /* Makes the link text all uppercase */
}
```

9. **:hover (Pseudo-Class)**

Defines styles for when the user hovers over the link. This is commonly used to create interactive effects.

Example: Change the link color when hovered.

```css
a:hover {
    color: #ff6347; /* Change text color to tomato */
    text-decoration: underline; /* Underline the text on hover */
}
```

10. **:active (Pseudo-Class)**

Defines the style of the link when it's being clicked (active state).

Example: Add a background color when the link is clicked.

```css
a:active {
    background-color: #007bff; /* Blue background on click */
}
```

11. **:visited (Pseudo-Class)**

Defines the style of a link that has already been visited.

Example: Change the color of visited links.

```css
a:visited {
    color: #800080; /* Purple color for visited links *
}
```

Pseudo Class:

A pseudo-class in CSS is a keyword added to selectors that specifies a special state of the element. They allow you to define how elements should look in specific conditions, such as when they are hovered over, clicked on, or visited. Pseudo-classes help you style elements dynamically based on user interaction or the state of the element, without needing to modify the HTML or add extra classes.

Common Pseudo-Classes:

- ☑ :hover – Used when the user hovers over an element.
- ☑ :active – Used when an element is being activated (e.g., clicked on).
- ☑ :visited – Used when a link has been clicked and visited by the user.
- ☑ :focus – Used when an element, such as an input field, receives focus (e.g., clicked into or tabbed into).
- ☑ :first-child – Used to select the first child element of a parent element.
- ☑ :last-child – Used to select the last child element of a parent element.
- ☑ :nth-child() – Used to select elements based on their position in a list of siblings.
- ☑ :not() – Used to select elements that do not match a given selector.

Example (Pseudo Class):

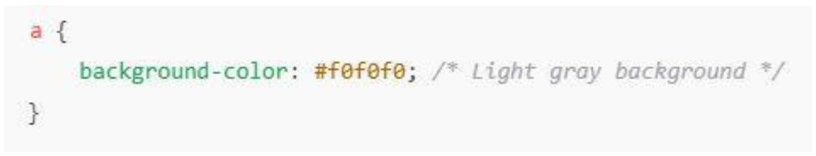
CSS Properties Related to Backgrounds:

Background properties in CSS allow you to set colors, images, positioning, and other characteristics of the background of an element (including the <a> tag).

1. **background-color**

   Sets the background color of an element.

   Example: Set the background color for the <a> tag.

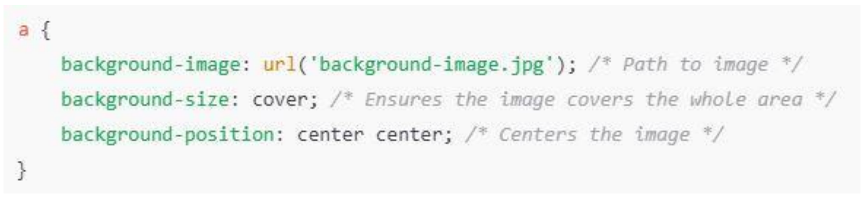


```css
a {
    background-color: #f0f0f0; /* Light gray background */
}
```

2. **background-image**

   Defines an image to be used as the background of an element.

   Example: Set a background image for the anchor tag.



```css
a {
    background-image: url('background-image.jpg'); /* Path to image */
    background-size: cover; /* Ensures the image covers the whole area */
    background-position: center center; /* Centers the image */
}
```

3. background-position

Sets the starting position of the background image.

Example: Position the background image to the top right of the anchor.

```
a {
    background-image: url('background-image.jpg');
    background-position: top right;
}
```

Normal Usage:

The background-position property in CSS is used to control the position of the background image within an element. It specifies where the background image should be placed relative to the element's content box. This property is particularly important when you're working with background images that are larger than the element or if you want to adjust how the image fits within the element.

Syntax:

```
background-position: <horizontal-position> <vertical-position>;
```

<horizontal-position>: Specifies the horizontal position of the background image.

It can be values like left, center, right, or a length/percentage (e.g., 20px, 50%).

<vertical-position>: Specifies the vertical position of the background image.

It can be values like top, center, bottom, or a length/percentage (e.g., 20px, 50%).

You can use one or both values, and if only one value is provided, the other will default to center.

Default Values:

- background-position: 0% 0%: This is the default value, meaning the background image starts at the top-left corner of the element.
- background-position: center: When a single value is provided, center will be used for both horizontal and vertical alignment.

Possible Values:

Keywords:

left, right, top, bottom, center: These keywords are used to position the background image at specific points relative to the element.

Length Values:

Can be in pixels (px), em units (em), rem units (rem), etc.

E.g., 20px, 50px, etc.

Percentage Values:

A percentage represents a relative position of the image with respect to the element's dimensions.

50% positions the background in the center of the element.

How it works?

The background-position property works by positioning the background image inside the element. You specify where the image should be positioned in relation to the top-left corner of the element. If the image is smaller than the element, the image is simply placed at the specified position. If the image is larger, parts of it may be cut off depending on the positioning.

Examples of background-position Usage:

- Using background-position with Keywords like keywords like top, center, bottom, left, and right.
- Using background-position with Length Values to specify exact positions using pixels,
- Using background-position with Percentage Values when you want a responsive design.
- Combining Horizontal and Vertical Values with both keywords and measurements to get a more customized background position.

4. background-repeat

Defines how the background image will repeat (e.g., repeat, no-repeat, repeat-x, repeat-y).

Example: Prevent the background image from repeating.

```
a {
    background-image: url('background-image.jpg');
    background-repeat: no-repeat; /* Prevents the image from repeating */
}
```

Normal Usage:

The background-repeat property in CSS is used to control how a background image is repeated (or tiled) within an element. This property is particularly useful when you want to repeat a background image horizontally, vertically, or in both directions, or when you don't want the image to repeat at all.

Syntax:

```
element {
    background-repeat: repeat | repeat-x | repeat-y | no-repeat;
}
```

Values for background-repeat:

a) repeat (default):

The background image will be repeated both horizontally and vertically until it covers the entire background area of the element.

b) repeat-x:

The background image will be repeated horizontally only (along the x-axis). It won't repeat vertically.

c) repeat-y:

The background image will be repeated vertically only (along the y-axis). It won't repeat horizontally.

d) no-repeat:

The background image will not be repeated. It will be displayed only once.

e) space (introduced in CSS3):

The background image will be repeated as many times as possible without stretching or clipping, and the remaining space will be distributed evenly between the images.

f) round (introduced in CSS3):

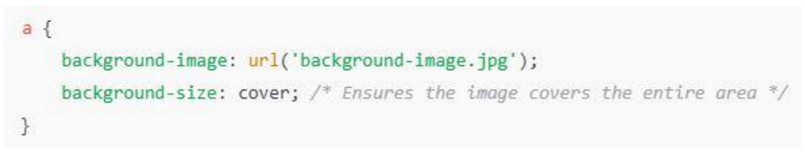
The background image will be stretched or shrunk, if necessary, to fit the entire area of the element without leaving any gaps.

Example:

5. background-size

Specifies the size of the background image (e.g., cover, contain, or specific pixel dimensions).

Example: Scale the background image to cover the entire anchor.

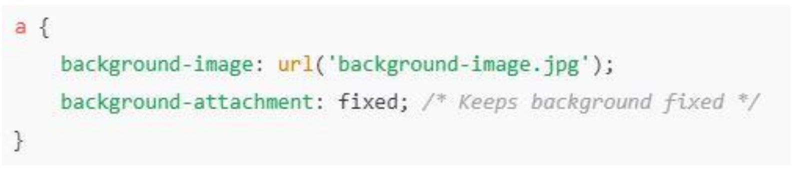


```
a {
    background-image: url('background-image.jpg');
    background-size: cover; /* Ensures the image covers the entire area */
}
```

6. background-attachment

Specifies whether the background image is fixed with regard to the viewport or scrolls with the content.

Example: Make the background image fixed so it doesn't scroll.

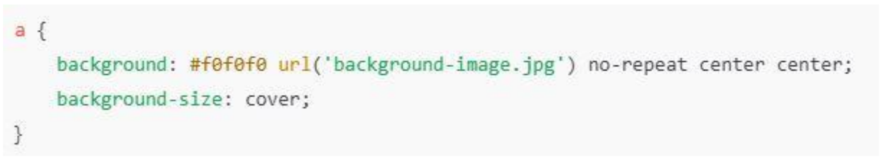


```
a {
    background-image: url('background-image.jpg');
    background-attachment: fixed; /* Keeps background fixed */
}
```

7. **background**

A shorthand property to set multiple background-related properties (like color, image, position, etc.) in one line.

Example: Combine background properties.



```
a {
    background: #f0f0f0 url('background-image.jpg') no-repeat center center;
    background-size: cover;
}
```

Margin, Padding & Border in CSS:

1. Margin:

The margin property creates space outside an element. It is used to control the space between the element's border and surrounding elements.

Sub-properties:

- ☑ **margin-top:** Specifies the margin space at the top of the element.
- ☑ **margin-right:** Specifies the margin space on the right side of the element.
- ☑ **margin-bottom:** Specifies the margin space at the bottom of the element.
- ☑ **margin-left:** Specifies the margin space on the left side of the element.
- ☑ **margin:** A shorthand for all four margins (top, right, bottom, left).

Example:

2. **Padding:**

The padding property creates space inside an element, between the content and the border. This affects the size of the element itself.

Sub-properties:

- ☑ **padding-top:** Specifies the padding space at the top of the element.
- ☑ **padding-right:** Specifies the padding space on the right side of the element.
- ☑ **padding-bottom:** Specifies the padding space at the bottom of the element.
- ☑ **padding-left:** Specifies the padding space on the left side of the element.
- ☑ **padding:** A shorthand for all four paddings (top, right, bottom, left).

Example:

3. **Border:**

The border property defines a visible boundary around an element. The border is placed inside the margin but outside the padding area.

Sub-properties:

- ☑ border-width: Specifies the width of the border.

- ☑ border-style: Specifies the style of the border (e.g., solid, dashed, dotted).
- ☑ border-color: Specifies the color of the border.
- ☑ border: A shorthand for border-width, border-style, and border-color.

Example:

Different ways to represent units in CSS:

In CSS, units define the measurement for properties such as width, height, padding, margin, font size, etc. These units can be broadly classified into two categories: relative units and absolute units.

1. Absolute Units

Absolute units are fixed in size and do not change based on other factors such as screen size or viewport. They are generally used for precise, fixed-size layouts or print styling.

Common Absolute Units:

- ☑ px (pixels)
- ☑ pt (points)
- ☑ cm (centimeters)
- ☑ mm (millimeters)
- ☑ in (inches)
- ☑ pc (picas)

Example:

2. Relative Units

Relative units are not fixed and depend on other factors like the parent element or the viewport size. They are ideal for responsive design as they scale based on different screen sizes or containers.

Common Relative Units:

- ☑ em
- ☑ rem
- ☑ % (percent)
- ☑ vw (viewport width)
- ☑ vh (viewport height)
- ☑ ch (character unit)
- ☑ vmin / vmax (viewport minimum/maximum)

Conclusion:

- ☑ Absolute units (e.g., px, pt, cm) are fixed and often used in print designs or precise layouts.

☑ Relative units (e.g., em, rem, %, vw, vh) are more flexible and responsive, adapting to changes in the environment (like font size, parent element, or viewport size).

Choosing the appropriate unit depends on the context of your design and whether you want fixed, scalable, or responsive measurements. For modern web design, relative units such as em, rem, %, vw, and vh are often favored for responsiveness.

Example:

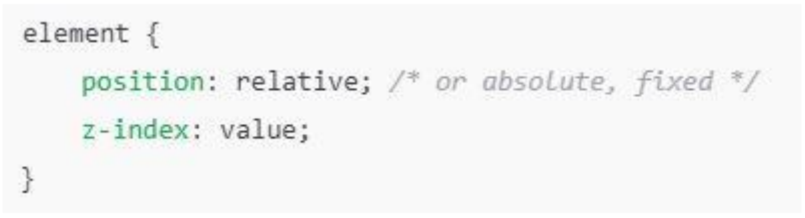
Demonstration of Z-Index value in CSS:

The z-index property in CSS is used to control the stacking order of elements on a webpage. It specifies the stacking order of elements that overlap with each other. The higher the z-index value, the closer the element is to the front (i.e., it will appear above elements with lower z-index values).

How Does z-index Work?

- The z-index only works on elements that have a position value other than static (which is the default).
- The z-index values can be positive, negative, or zero.
- Elements with higher z-index values will appear in front of those with lower values.

Syntax:



```css
element {
    position: relative; /* or absolute, fixed */
    z-index: value;
}
```

Valid Values for z-index:

- **auto:** The default value. The element is stacked according to its position in the document flow.
- **Integer values:** Positive, zero, or negative values (e.g., 1, -1, 100, 0).
- **inherit:** The z-index value will be inherited from the parent element.

Example1:

Example2:

CSS properties related to lists:

1. **Unordered List (<ul>):** An unordered list is typically used when the order of the list items doesn't matter, and each list item is marked with a bullet (or another marker).

   Example:

2. **Ordered List (<ol>):** An ordered list is used when the order of items matters (e.g., steps in a process). It is usually numbered by default.

   Example:

3. **Description List (<dl>):** A description list is used for defining terms and their descriptions. It consists of <dt> (definition term) and <dd> (definition description) elements.

   Example:

Illustration of CSS display property:

In CSS, the display property is used to define how an element should be displayed on the page and how it affects the layout of surrounding elements. It plays a crucial role in controlling the flow of content, and depending on its value, it determines whether an element is treated as a block, inline, or a more specialized layout type (e.g., flex or grid).

Common display Values and Their Effects:

- **display: block:**

  The element takes up the full width available, and each element appears on a new line.

  Examples: <div>, <p>, <h1>, <section>, etc.
- **display: inline:**

  The element takes up only as much width as its content requires and does not cause a line break.

  Examples: <span>, <a>, <strong>, etc.
- **display: inline-block:**

  The element behaves like an inline element (doesn't create a new line) but allows you to set width and height like a block element.
- **display: none:**

  The element will not be displayed at all, and it is removed from the document flow.
- **display: flex:**

  A modern layout model that allows flexible, responsive layouts by aligning items in a row or column.

  The container element becomes a flex container, and its children become flex items.
- **display: grid:**

  A 2D layout model for creating complex grid-based designs, where both rows and columns are defined.

Example:

Custom Cursors:

Custom cursors are a way to change the default pointer behavior in web applications or websites. By default, a browser provides a standard set of cursors (e.g., pointer, text, crosshair), but CSS allows developers to use images or vector graphics to replace these default cursors.

Example-1

Example-2

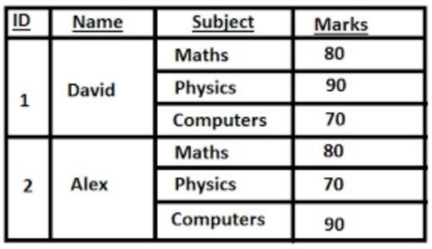
LAB PROGRAMS

Week 1: HTML Basic Tags

1. Write an HTML program using header tags `<h1>` to `<h6>`.
2. Write an HTML program using the paragraph tag `<p>`.
3. Write an HTML program using the anchor tag `<a>`.
4. Write an HTML program using the image tag `<img>` with all its attributes.
   - ☑ Program-1
   - ☑ Program-2
5. Write an HTML program using the marquee tag `<marquee>`.
6. Create a basic homepage of a university using HTML.

Week 2: HTML Lists and Tables

1. Create unordered, ordered, and description lists using `<ul>`, `<ol>`, and `<dl>` tags.
2. Write an HTML program to describe all types of lists along with their tags and attributes.
3. Design an HTML table using `rowspan` and `colspan`.



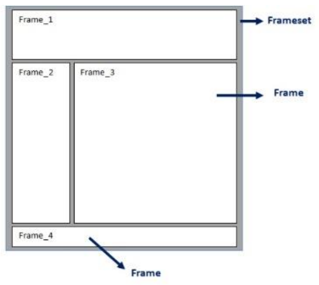
ID	Name	Subject	Marks
1	David	Maths	80
		Physics	90
		Computers	70
2	Alex	Maths	80
		Physics	70
		Computers	90

Design a HTML Table in the above Given Format?

4. Create a timetable using HTML (Table tags and their attributes).
   - ☑ Program-1
   - ☑ Program-2
5. Demonstrate the attributes of `<table>`, `<tr>`, and `<td>` tags with examples.

Week 3: HTML Frames and Forms (with Validations)

1. **Create an HTML page using `<frameset>` and `<frame>` tags to divide the browser window into multiple sections**



Design a HTML Frame in the above given Format?

2. **Write an HTML program to demonstrate target frames and attributes of `<frameset>`.**
3. **Write an HTML program to print a flag using `<frameset>`.**
4. **Design an HTML Gmail login form using `<form>` tags.**
5. **Create a registration form for personal details in HTML.**
6. **Create a university application form for applying for an M.Tech course, collecting details like Name, Highest Degree, Gender, GATE Rank, courses applied for (list), and interests (checkboxes), with a submit button leading to a Thank You page**

Week 4: CSS Styles, Properties, and Pseudo-classes

1. **Create a webpage that demonstrates the use of inline styles, internal stylesheets, and external stylesheets, including CSS properties related to text, font, background, tables, and lists.**
   - ☑ **Inline CSS**
   - ☑ **Internal CSS**
   - ☑ **External CSS**
2. **Create a webpage demonstrating the use of different selectors in CSS, including properties related to margins, padding, and borders.**
3. **Create a webpage demonstrating the effects of different pseudo-classes on anchor tags (e.g., :hover, :visited, :active).**