Unit-1

- **WEB ESSENTIALS AND STYLE SHEETS**
- Clients, Servers, and Communication.
- The Internet - Basic Internet Protocols –
- The World WideWeb-HTTPrequestmessage-responsemessage-WebClients-WebServers-
- Markup Languages: XHTML. An Introduction to HTML – History –Versions - Basic XHTML Syntax and Semantics - Fundamentals of HTML.
- CSS-IntroductiontoCascadingStyleSheets–Features-CoreSyntax-StyleSheetsand HTML - Cascading and Inheritance - Text Properties – Positioning.

- **Client**
- A **client** is a computer/device or software that **requests services** from another computer (the server).
- Clients usually initiate the communication.
- Examples:
  - Web browser (Google Chrome, Firefox, Edge).
  - Mobile apps (Instagram, WhatsApp).
  - Email clients (Outlook, Gmail app).
- Clients are **service requesters**.

- **Server**
- A **server** is a computer/system that **provides services or resources** to clients.
- Servers are usually powerful machines with specialized software.
- Examples:
  - **Web Server** (Apache, Nginx) → serves web pages.
  - **Database Server** (MySQL, Oracle) → stores and provides data.
  - **Mail Server** → handles emails.
- Servers are **service providers**.

- **Client–Server Communication**
- Based on the **Client–Server model** of computing.
- **Steps in communication**:
  - Client sends a **request** (e.g., HTTP GET request).
  - Server **processes** the request.
  - Server sends a **response** (e.g., web page, data, or error message).
- Communication happens over a **network** (usually Internet or LAN).
- Protocols define **how communication happens**:
  - HTTP/HTTPS (for web browsing).
  - FTP (for file transfer).
  - SMTP/IMAP (for email).

- **Examples of Client–Server Communication**
- Typing www.wikipedia.org in a browser:
  - **Client (browser)** → sends HTTP request.
  - **Server (Wikipedia's web server)** → sends back the web page.
- Sending a WhatsApp message:
  - **Client (your phone app)** → sends message to WhatsApp server.
  - **Server** → forwards it to recipient's client app.

- **Examples of Client–Server Communication**
- Typing www.wikipedia.org in a browser:
  - **Client (browser)** → sends HTTP request.
  - **Server (Wikipedia's web server)** → sends back the web page.
- Sending a WhatsApp message:
  - **Client (your phone app)** → sends message to WhatsApp server.
  - **Server** → forwards it to recipient's client app.

- **Advantages of Client–Server Model**
- Centralized resources (data stored on servers).
- Easier maintenance (update server, all clients benefit).
- Scalability (many clients can connect).
- Security (controlled access to resources).

- **Client** = Requests services (browsers, apps).
- **Server** = Provides services (web, database, mail).
- **Communication** = Clients send requests → Servers process and respond.

The Internet - Basic Internet Protocols

- **The Internet**
- The **Internet** is a global network of interconnected computers that communicate with each other using standardized protocols.
- It allows sharing of information, communication, and access to resources.
- Services provided by the Internet:
  - **World Wide Web (WWW)** → Websites and browsing.
  - **Email** → Communication using mail servers.
  - **File Transfer** → Uploading/downloading files.
  - **Remote Access** → Accessing systems from anywhere.
- Internet as a **network of networks**.

- **What is a Protocol?**
- A **protocol** is a set of rules that defines **how computers communicate** over a network.
- Without protocols, devices would not understand each other.

- **. Basic Internet Protocols**

a) IP (Internet Protocol)

The backbone of the Internet.

Responsible for addressing and routing data from source to destination.

Each device has a unique IP address (IPv4: 192.168.1.1, IPv6: 2001:db8::1).

(b) TCP (Transmission Control Protocol)

Works with IP → Together called TCP/IP.

Provides reliable communication (ensures no data is lost).

Breaks data into packets, ensures they arrive in the correct order.

(c) UDP (User Datagram Protocol)

Faster, lightweight alternative to TCP.

Used when speed matters more than reliability.

Examples: Online gaming, video streaming, VoIP (calls).

(d) HTTP / HTTPS (Hypertext Transfer Protocol / Secure)

Used for web browsing.

HTTP → Transfers webpages between client (browser) and server.

HTTPS → Encrypted version (secure, uses SSL/TLS).

(e) FTP (File Transfer Protocol)

Used to upload and download files between client and server.

Example: Transferring website files to a hosting server.

(f) SMTP, POP3, IMAP (Email Protocols)

SMTP (Simple Mail Transfer Protocol) → Sending emails.

POP3 (Post Office Protocol v3) → Downloading emails (removes from server).

IMAP (Internet Message Access Protocol) → Accessing emails while keeping them on the server.

(f) SMTP, POP3, IMAP (Email Protocols)

SMTP (Simple Mail Transfer Protocol) → Sending emails.

POP3 (Post Office Protocol v3) → Downloading emails (removes from server).

IMAP (Internet Message Access Protocol) → Accessing emails while keeping them on the server.

(g) DNS (Domain Name System)

Converts domain names (like www.google.com) into IP addresses.

Works like a phonebook of the Internet.

- **Quick Example of Protocols Working Together**
- When you visit https://www.wikipedia.org:
- **DNS** translates domain name → IP address.
- **IP** locates the server.
- **TCP** establishes a reliable connection.
- **HTTPS** transfers the webpage securely.

- **World Wide Web (WWW)**

- The **World Wide Web (WWW)** is a system of interlinked hypertext documents accessible via the Internet.
- Uses **web browsers** (Chrome, Firefox, Edge, etc.) to view and navigate content.
- Documents are written in **HTML** (Hypertext Markup Language).
- Each resource on the web has a unique address called a **URL** (Uniform Resource Locator).
- Works on the **Client–Server model**.
- Example: When you type www.google.com, your browser (client) sends a request to Google’s server, which responds with the webpage.

- **HTTP (Hypertext Transfer Protocol)**
- HTTP is the **communication protocol** used between **web clients** and **web servers**.
- It is **stateless**, meaning each request is independent.
- Runs mostly over **TCP/IP** (Transmission Control Protocol / Internet Protocol).

- **HTTP Request Message**
- When a client (browser) wants data, it sends an HTTP Request.
- Structure of a Request:
- 1. Request Line → Method, URL, HTTP version
- Example: GET /index.html HTTP/1.1
- 2. Header Fields → Provide additional information (Host, User-Agent, Cookies, etc.)
- 3. Body (optional) → Data sent to server (used in POST, PUT methods).
- Common HTTP Methods:
- • GET → Request data (read only).
- • POST → Send data (e.g., form submission).
- • PUT → Update data.
- • DELETE → Remove data.

- **HTTP Response Message**
- After processing a request, the server sends back an HTTP Response.
- Structure of a Response:
- 1. Status Line → Protocol version, Status code, Reason
- o Example: HTTP/1.1 200 OK
- 2. Header Fields → Information about server, content type, length, etc.
- 3. Body → The actual content (HTML, image, video, JSON, etc.).
- Common Status Codes:
- 200 OK → Success.
- 301 Moved Permanently → Resource moved to a new URL.
- 404 Not Found → Resource not found.
- 500 Internal Server Error → Server issue.

- **. Web Clients**
- Software that **sends requests** to web servers and **displays responses**.
- Example: Web browsers (Chrome, Firefox, Safari, Edge).
- Functions:
  - Render HTML pages.
  - Execute scripts (JavaScript).
  - Store cookies, cache, and session data.

- **Web Servers**
- Software that **receives HTTP requests** from clients and **sends back responses**.
- Example: Apache, Nginx, Microsoft IIS.
- Functions:
  - Host and serve web content (HTML, CSS, JS, images, videos).
  - Handle client requests using HTTP/HTTPS.
  - Manage resources and security.

- **Client–Server Interaction Example**
- User enters http://example.com in browser.
- Browser (client) sends **HTTP GET request** to server.
- Server finds the requested file (index.html) and sends it back as **HTTP Response**.
- Browser receives response and displays the webpage

- **Introduction to Markup Languages**
- A **markup language** is a computer language used to **structure, describe, and format documents**.
- It uses **tags** (e.g., <p>, <h1>, <img>) to define how content should be displayed in a web browser.
- Examples: **HTML, XHTML, XML, Markdown**.

- **. HTML (Hypertext Markup Language)**
- HTML is the **standard language for creating web pages**.
- It defines the **structure of web documents** using a system of tags.
- A basic HTML document:
- <!DOCTYPE html>
- <html>
- <head>
- <title>My First Page</title>
- </head>
- <body>
- <h1>Hello, World!</h1>
- This is my first web page.
- </body>

- **. History of HTML**
- **1989–1991** → Tim Berners-Lee invented HTML as part of the World Wide Web project.
- **1993** → HTML 1.0 (first release).
- **1995** → HTML 2.0 (standardized by IETF).
- **1997–1999** → HTML 3.2 and HTML 4.0 introduced more features (tables, styles, scripts).
- **2000** → XHTML 1.0 (stricter, XML-based HTML).
- **2014 onwards** → HTML5 became the current standard (supports audio, video, canvas, semantic tags).

- **Versions of HTML**
- **HTML 1.0** → Basic text formatting, hyperlinks.
- **HTML 2.0** → Forms, images, tables.
- **HTML 3.2** → Support for scripting, applets, styles.
- **HTML 4.0/4.01** → Frames, CSS support, scripting improvements.
- **XHTML 1.0** → HTML reformulated as XML (stricter rules).
- **HTML5** → Multimedia support, semantic elements, mobile-friendly.

- **XHTML (Extensible Hypertext Markup Language)**
- XHTML = **HTML written as well-formed XML**.
- Developed by W3C to make HTML more consistent and strict.
- Benefits:
  - Cleaner, well-structured code.
  - Easy to integrate with XML-based technologies.
  - More reliable for browsers and mobile devices.

- Basic XHTML Syntax and Semantics
- XHTML follows stricter rules than HTML:
- **Syntax Rules in XHTML**
- 1. **Tags must be properly nested**
- 2. Text <!-- Correct -->
- 3. Text <!-- Wrong -->
- 4. **All tags must be closed**
- 5.
 <!-- Correct in XHTML -->
- 6.
 <!-- Wrong -->
- 7. **Lowercase for all tags and attributes**
- 8. Correct
- 9. <P>Wrong</P>
- 10. Attribute values must be in quotes
- 11.
- 12. **Root element <html> must include XML namespace**
- 13. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
- 14. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
- 15. <html xmlns="http://www.w3.org/1999/xhtml">
- 16. <head><title>Example</title></head>
- 17. <body> Hello XHTML! </body>
- 18. </html>

- **Semantics in XHTML**
- **Semantics** = meaning of tags.
- Tags should be used for their **intended purpose**:
  - <h1> … </h1> → Main heading.
  - … → Paragraph.
  - … → Emphasized text (instead of <i> for italics).
  - … → Important text (instead of <b> for bold).

- **HTML** = standard language of the web.
- **XHTML** = stricter, XML-based version of HTML.
- **HTML History** = evolved from simple text links → modern multimedia pages (HTML5).
- **XHTML Syntax** = lowercase tags, all attributes in quotes, all tags closed, proper nesting.
- **Semantics** = tags should carry meaning, not just appearance.

Feature	HTML	XHTML
Definition	Hypertext Markup Language – standard language for creating web pages.	Extensible Hypertext Markup Language – stricter, XML-based version of HTML.
Syntax rules	More flexible, browsers often “forgive” mistakes.	Very strict – must follow XML rules.
Tag Case	Tags can be uppercase or lowercase.	Tags must be lowercase only.
Closing Tags	Some tags may remain unclosed (e.g.,).	All tags must be closed properly (e.g.,).
Attribute Values	Quotes are optional (<input type=text>).	Quotes are mandatory (<input type="text" />).
Empty Elements	Can be written without / (e.g., <img>).	Must be written with / (e.g., <img />).
Error Handling	Browsers try to correct mistakes automatically.	No error tolerance – incorrect code may not display.
Document Structure	Doctype is optional.	Doctype declaration

- CSS (Cascading Style Sheets) is a style sheet language used to describe the presentation of HTML documents. It controls the layout, colors, fonts, spacing, and overall visual appearance of web pages.

- Why Use CSS?
- Separates content (HTML) from presentation (CSS)

- Reusability across multiple pages

- Easier maintenance and cleaner HTML

- Enables responsive and visually appealing designs

- **Features of CSS**
- **Separation of content and style – HTML handles structure, CSS handles design.**
- **Improves presentation – Fonts, colors, spacing, layout, animations.**
- **Reusability – A single CSS file can style multiple web pages.**
- **Efficiency – Reduces code repetition.**
- **Responsive Design – Adjusts layout for different screen sizes.**
- **Global consistency – Same look and feel across a website.**

- **Core CSS Syntax**
- **CSS has a simple rule-based syntax:**
- **selector {**
- **property: value;**
- **}**
- **Selector → Identifies which HTML element(s) to style.**
- **Property → The style feature to change (e.g., color, font-size).**
- **Value → The setting for the property.**
- **Example:**
- **p {**
- **color: blue;**
- **font-size: 16px;**
- **}**
- **This makes all <p> elements blue and sets font size to 16px.**

- Types of CSS
- 1. Inline CSS
- 2. Internal CSS (Embedded CSS)
- 3. External CSS

1. Inline CSS

- CSS is written directly in the HTML tag using the style attribute.

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

- Advantages:
- Quick and easy for small changes
- Useful for testing/debugging

- Disadvantages:
- Not reusable
- Clutters HTML code

2. Internal CSS (Embedded CSS)

- CSS is written inside a <style> tag within the <head> of the HTML document.
- <!DOCTYPE html>
- <html>
- <head>
- <style>
- h1 {
- color: green;
- font-size: 30px;
- }
- </style>
- </head>
- <body>
- <h1>Welcome</h1>
- </body>
- </html>

- Advantages:
- Keeps styles in one place within the same file
- Good for small projects or single pages

- Disadvantages:
- Not reusable across multiple HTML files
- Makes HTML file heavier

3. External CSS

- CSS is written in a separate file with a .css extension and linked using the <link> tag in the <head> section.
- Syntax:
- HTML:
- <link rel="stylesheet" href="styles.css">
- styles.css:
- css
- h1 {
- color: red;
- font-size: 36px;
- }

- Advantages:
- Reusable across multiple HTML pages
- Keeps HTML clean
- Better maintainability for large projects

- Disadvantages:
- Requires additional HTTP request to load the stylesheet

Inline → Quick but messy.

Internal → Good for single-page styling.

External → Best practice for professional projects.

- **Cascading in CSS**
- The word *Cascading* means that when **multiple style rules** apply to the same element, the browser decides **which one takes priority**
- **Inline Styles** (highest).
- **Internal Styles**.
- **External Styles**.
- **Browser Defaults** (lowest).

- cell padding and cell spacing—two important concepts in HTML tables that affect the spacing inside and between table cells.
- 1. Cell Padding
- It is the space between the cell content and the cell border.
- It controls the inner spacing of the table cells.
- Example:
- Each cell will have **10px padding** inside (between content and border).<table border="1" cellpadding="10">
- <tr>
- <td>Cell 1</td>
- <td>Cell 2</td>
- </tr>
- </table>
- Each cell will have **10px padding** inside (between content and border).

- In CSS, you can also do this with:

- td {
- padding: 10px;
- }

- Cell Spacing
- It is the space between adjacent table cells.
- It controls the outer spacing between cells.

- Example:
- <table border="1" cellspacing="10">
- <tr>
- <td>Cell 1</td>
- <td>Cell 2</td>
- </tr>
- </table>
- There will be 10px space between the cells.

- In CSS, use border-spacing:

- table {
- border-spacing: 10px;
- }

Feature	Description	HTML Attribute	CSS Property
Cell Padding	Space **inside** the cell (content to border)	cellpadding="value"	padding (on td)
Cell Spacing	Space **between** the cells (border to border)	cellspacing="value"	border-spacing (on table)

- What is an HTML Form?
- An HTML form is used to collect user input and send it to a server for processing (e.g., login, registration, feedback).

- It uses the <form> tag, and inside it, we place form controls like text fields, checkboxes, radio buttons, and submit buttons.

- Basic Syntax
- <form action="submit_form.php" method="post">
- <!-- Form elements go here -->
- </form>

Attribute	Description
action	The URL where form data is sent for processing
method	HTTP method (GET or POST)
name	Name of the form
target	Where to display the response (_self, _blank, etc.)

Common Form Elements:

- 1. Text Input
- <label for="name">Name:</label>
- <input type="text" id="name" name="username">
- 2. Password Field
- <label for="password">Password:</label>
- <input type="password" id="password" name="pass">
- 3. Radio Buttons
- Gender:
- <input type="radio" id="male" name="gender" value="Male">
- <label for="male">Male</label>
- <input type="radio" id="female" name="gender" value="Female">
- <label for="female">Female</label>

- 4. Checkboxes
- Hobbies:
- <input type="checkbox" id="reading" name="hobby" value="Reading">
- <label for="reading">Reading</label>
- <input type="checkbox" id="traveling" name="hobby" value="Traveling">
- <label for="traveling">Traveling</label>

- 5. Dropdown (Select Menu)
- <label for="country">Country:</label>
- <select id="country" name="country">
- <option value="India">India</option>
- <option value="USA">USA</option>
- <option value="UK">UK</option>
- </select>

- 6. Textarea
- <label for="message">Message:</label>
- <textarea id="message" name="message" rows="4" cols="30"></textarea>
- 7. Submit Button
- <input type="submit" value="Submit">

- 8. Reset Button
- <input type="reset" value="Clear">
- Form Methods
- GET → Sends form data in the URL (used for search forms, not secure)

- POST → Sends form data in the request body (used for login, registration, secure data)

- <form action="process.php" method="post">
- <!-- form elements -->
- </form>

```
<!DOCTYPE html>
<html>
<body>
  <h2>Registration Form</h2>
  <form action="submit.php" method="post">
    <label for="fname">First Name:</label>
    <input type="text" id="fname" name="firstname"><br><br>

    <label for="lname">Last Name:</label>
    <input type="text" id="lname" name="lastname"><br><br>

    <label>Gender:</label>
    <input type="radio" name="gender" value="Male"> Male
    <input type="radio" name="gender" value="Female"> Female<br><br>

    <label for="country">Country:</label>
    <select id="country" name="country">
      <option value="India">India</option>
      <option value="USA">USA</option>
    </select><br><br>

    <input type="submit" value="Register">
    <input type="reset" value="Clear">
  </form>
</body>
</html>
```

- Cascading in CSS
- Definition: CSS stands for Cascading Style Sheets. “Cascading” means styles are applied in order of importance and conflict resolution.
- Order of Precedence (from lowest to highest):
- Browser default styles
- External stylesheet
- Internal stylesheet (inside <style> tag in HTML)
- Inline styles (inside the element’s style attribute)
- !important declaration (highest priority)

- Hello
- Inline red will override external CSS p { color: blue; }.

- Inheritance in CSS
- Definition: Child elements can inherit property values from their parent elements.
- Inheritable Properties: font, color, line-height, visibility, text-align.
- Non-inheritable Properties: margin, padding, border, background.

-
- This text is green (inherited).
- This is also green.
-

- s — one with color: green and one without

- **Text Properties in CSS**
- Text properties control appearance and spacing of text.

- **Positioning in CSS**
- Defines how an element is placed on the web page.
- **Static (default):**
  Elements appear in normal document flow.
- p { position: static; }

- **Relative:**
  Positioned relative to its normal position.
- p { position: relative; top: 20px; left: 30px; }
- **Absolute**:
- Positioned relative to the nearest positioned ancestor (not the page).
- div { position: absolute; top: 50px; left: 50px; }
- **Fixed:**
  Stays fixed on screen even when scrolling.
- nav { position: fixed; top: 0; }

- **Sticky:**
  Acts like relative until scrolling reaches a threshold, then “sticks”.
- header { position: sticky; top: 0; }