UNIT 1: WEB ESSENTIALS

Clients and Servers

- **Client:** A hardware device or software application that requests services or resources from a server. Clients include web browsers, mobile apps, or other interfaces that communicate with a web server over the internet.
- **Server:** A powerful computer system or application that stores, processes, and manages network resources and services. Web servers handle client requests by serving web pages, running applications, and managing databases.

Communication

- The **Client-Server Model** is the foundation of internet communication.
- Communication happens over protocols like HTTP, where clients send requests and servers respond accordingly.
- **Stateless Communication:** HTTP is a stateless protocol, meaning each request is independent of previous requests unless sessions or tokens are used.

Basic Internet Protocols

- **IP (Internet Protocol):** Assigns unique addresses to devices and handles routing.
- **TCP (Transmission Control Protocol):** Manages data packet transmission and ensures order and integrity.
- **HTTP (HyperText Transfer Protocol):** Primary protocol for web communication; operates over TCP.
- **HTTPS (HTTP Secure):** Adds encryption using SSL/TLS.
- **DNS (Domain Name System):** Resolves human-readable domain names (e.g., google.com) to IP addresses.
- **FTP (File Transfer Protocol):** Transfers files to/from servers; often used for uploading websites.

HTTP Request Message

Structure and Explanation:

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

- **Request Line:** Specifies the method (GET), resource, and HTTP version.
- **Headers:** Provide information about the client and request parameters.
- **Body:** Included in methods like POST, contains form or JSON data.

HTTP Response Message

Structure and Explanation:

```
HTTP/1.1 200 OK
Content-Type: text/html
```

```
<html><body>Welcome</body></html>
```

- **Status Code:** Indicates the result of the request (200 OK, 404 Not Found, 500 Internal Server Error).
- **Headers:** Include metadata (e.g., content type, length).
- **Body:** Actual response data (HTML, JSON, etc.).

Web Clients

- Any software capable of making HTTP requests and displaying/handling server responses.
- Examples:
    - Web Browsers: Chrome, Edge, Firefox
    - Mobile Applications: Android/iOS apps
    - REST Clients: Postman, Curl

Generations of Web Applications

1. **Web 1.0:** Static HTML pages, limited user interaction.
2. **Web 2.0:** Dynamic content using JavaScript, AJAX, rich user interfaces, and user-generated content.
3. **Web 3.0:** Emphasizes semantic data, artificial intelligence, and decentralized networks (blockchain, peer-to-peer apps).

Web Server Configuration

- Includes configuring server software to respond to web requests.
- Tasks involve:
    - Setting up **Virtual Hosts**
    - Enabling **SSL Certificates**
    - Defining **Root Directory** and error pages
    - Managing access and security settings
- Tools: Apache configuration (`httpd.conf`), Nginx (`nginx.conf`)

Debugging Tools: Postman

- **Postman** is a GUI-based tool for testing and debugging REST APIs.
- Features:
    - Craft HTTP requests with parameters, headers, and bodies
    - Inspect detailed response data
    - Save and organize API collections
    - Simulate various environments using variables

1. Clients and Servers

Client

- A client is a computer or device that sends requests to a server.
- Examples: Web browsers (Chrome, Firefox), mobile apps.

Server

- A server is a powerful computer that listens for requests and sends responses.
- Examples: Apache, Nginx, Node.js servers.

Client-Server Communication Flow

```
Client (Browser) -----> Request (HTTP) -----> Server
         <----- Response (HTML/JSON) <-----
```

2. Communication Model

- **Request/Response model** (stateless)
- Uses **HTTP or HTTPS**
- Typical flow:
    1. Client makes an HTTP request (e.g., GET/POST).
    2. Server processes the request.
    3. Server sends an HTTP response (HTML/JSON).

3. Basic Internet Protocols

Protocols:

Protocol	Description
IP	Internet Protocol - provides IP addresses
TCP	Transmission Control Protocol - ensures reliable transmission
UDP	User Datagram Protocol - faster but less reliable
HTTP/HTTPS	Web data transmission protocols
FTP	File Transfer Protocol
SMTP/POP3/IMAP	Email communication

4. HTTP Request Message Structure

Example: `GET` request

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

Key Parts:

- **Request Line**: `GET /index.html HTTP/1.1`
- **Headers**: Information about client and request
- **Body** (only in POST, PUT): Sent data (e.g., form data)

5. HTTP Response Message Structure

Example: Response to a GET request

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 123

<html>
  <body>Hello, World!</body>
</html>
```

Key Parts:

- **Status Line**: `HTTP/1.1 200 OK`
- **Headers**: Content-Type, Content-Length
- **Body**: Actual web content (HTML, JSON, etc.)

6. Web Clients

- Software that communicates with web servers.
- Examples:
  - **Web Browsers**: Chrome, Firefox
  - **Mobile apps**
  - **Command-line tools**: `curl`, `wget`
  - **API Testing Tools**: Postman

7. Generations of Web Applications

Generation	Description	Example
Web 1.0	Static content	Simple HTML pages
Web 2.0	Dynamic, user interaction	Facebook, Gmail
Web 3.0	Semantic web, AI-powered	ChatGPT, personalized search
Web 4.0	Intelligent agents, IoT integration	Smart assistants, home automation

8. Web Server Configuration

Popular Servers:

- **Apache HTTP Server**
- **Nginx**
- **Node.js (Express)**

Basic Apache config example:

```
<VirtualHost *:80>
  ServerAdmin webmaster@example.com
  DocumentRoot "/var/www/html"
  ServerName www.example.com
  ErrorLog ${APACHE_LOG_DIR}/error.log
</VirtualHost>
```

Node.js Server Example:

```
const http = require('http');
http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello from Node.js Server');
}).listen(3000);
```

9. Debugging Tools: Postman

Postman Features:

- GUI tool to test HTTP APIs
- Supports methods: GET, POST, PUT, DELETE
- Send headers, body, authorization
- View response status, headers, body
- Create collections for test cases

Example: Testing a POST request in Postman

- **URL**: `https://api.example.com/login`
- **Method**: `POST`
- **Headers**: `Content-Type: application/json`
- **Body**:

```
{
  "username": "user1",
  "password": "pass123"
}
```

- **Response**:

```
{
  "token": "abc123"
}
```

UNIT 2: HTML / XHTML

HTML Overview

- HTML defines the structure of web content using **tags**.
- Each tag represents a type of element (e.g., heading, paragraph, image).
- Tags usually come in pairs: opening `<tag>` and closing `</tag>`.

Fundamental HTML Elements

- Structural Tags: `<html>`, `<head>`, `<body>`
- Metadata Tags: `<meta>`, `<title>`, `<link>`
- Content Tags:
  - Headings: `<h1>` to `<h6>`
  - Paragraph: `<p>`
  - Anchor: `<a href="url">link</a>`
  - Image: `<img src="img.jpg" alt="desc">`
  - Lists: `<ul>`, `<ol>`, `<li>`
  - Tables: `<table>`, `<tr>`, `<td>`, `<th>`
  - Forms: `<form>`, `<input>`, `<label>`, `<textarea>`

XHTML Syntax and Semantics

- XHTML combines HTML with XML's strict syntax rules:
  - Elements must be properly nested and closed.
  - Attribute values must be quoted.
  - Tag names must be in lowercase.
- Benefits: Better structure, easier parsing by machines.

Document Publishing

- Write HTML code and save with `.html` extension.
- Upload to a web server using FTP or hosting services.
- Use a browser to access via URL.

1. Markup Languages Overview

What is a Markup Language?

- A **markup language** is used to define the structure and presentation of text.
- Tags are used to annotate content.

Common Markup Languages:

- **HTML**: HyperText Markup Language – used to create web pages.
- **XHTML**: eXtensible HTML – stricter version of HTML, based on XML.
- **XML**: eXtensible Markup Language – used for data storage and transport.

2. HTML (HyperText Markup Language)

HTML Structure

Every HTML document has:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Page</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

Key HTML Elements:

Element	Description
`<html>`	Root of the HTML document
`<head>`	Metadata, styles, scripts
`<title>`	Title in the browser tab
`<body>`	Main content of the page
`<h1> to <h6>`	Headings
`<p>`	Paragraph
`<a>`	Hyperlink
`<img>`	Image
`<div>`	Generic container
`<span>`	Inline container

3. Basic XHTML Syntax and Semantics

Rules of XHTML (Stricter than HTML):

- All tags **must be closed**.
- Tags must be **properly nested**.
- Attribute values must be **quoted**.
- All elements must be **in lowercase**.

XHTML Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>XHTML Page</title>
  </head>
```

```
  <body>
    <h1>Welcome</h1>
    <p>This is valid XHTML.</p>
    <br />
  </body>
</html>
```

4. Document Publishing

Document Publishing in Web Development:

- **Save HTML/XHTML file** with `.html` or `.xhtml` extension.
- Open in a browser or upload to a **web server**.
- Tools:
    - Local: VS Code + Live Server
    - Online: GitHub Pages, Netlify, etc.

5. HTML4 vs HTML5

Feature	HTML4	HTML5
Doctype	Complex	Simple (`<!DOCTYPE html>`)
Syntax	Based on SGML	Looser, not SGML
Multimedia	No native audio/video	`<audio>`, `<video>` supported
New Elements	No semantic elements	Yes: `<section>`, `<article>`, `<nav>`, etc.
Form Input Types	Basic only	New types: `email`, `date`, `range`, etc.
APIs	Limited	Rich: Canvas, Geolocation, Drag-n-drop, Web Storage

HTML5 Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML5 Demo</title>
  </head>
  <body>
    <header>
      <h1>Welcome to HTML5</h1>
    </header>
    <section>
      <article>
        <p>This is a modern semantic layout.</p>
      </article>
    </section>
    <footer>
      <p>© 2025</p>
    </footer>
  </body>
</html>
```

UNIT 3: CSS3

Introduction

- CSS defines how HTML content is displayed (layout, colors, fonts).
- CSS3 introduces modular features (selectors, media queries, animations).

CSS Syntax

```
selector {
  property: value;
}
```

- Example:

```
h1 {
  color: blue;
  font-size: 24px;
}
```

Common Style Properties

- **Text Styles:** `font-family`, `font-size`, `font-weight`, `line-height`, `text-align`
- **Box Model:** Content > Padding > Border > Margin
- **Layout:** `display: block|inline|flex|grid`, `float`, `clear`
- **Positioning:** `static`, `relative`, `absolute`, `fixed`, `sticky`
- **Background:** `background-color`, `background-image`, `background-size`
- **Borders:** `border`, `border-radius`
- **Lists:** `list-style-type`, `list-style-position`
- **Tables:** `border-collapse`, `table-layout`
- **Cursor:** `cursor: pointer`, `cursor: not-allowed`

CSS Selectors

- **Basic Selectors:**
  - Element: `div`, `p`
  - Class: `.highlight`
  - ID: `#header`
- **Combinators:**
  - Descendant: `div p`
  - Child: `ul > li`
  - Adjacent sibling: `h1 + p`
- **Pseudo-selectors:**
  - `:first-child`, `:last-child`, `:hover`, `:nth-child()`
  - `::before`, `::after`, `::first-line`
- **Attribute Selectors:**
  - `input[type="text"]`, `a[target="_blank"]`
- **Group Selectors:**
  - `h1, h2, p` styles multiple elements at once

1. CSS3 Introduction
What is CSS3?

- **CSS** = Cascading Style Sheets – used to style HTML elements.
- **CSS3** is the latest version of CSS with new features like animations, transitions, and flexible layouts.

2. Features of CSS3

- **Media Queries** – for responsive designs.
- **Flexbox and Grid** – modern layout systems.
- **New Selectors** – nth-child, attribute, ::before, etc.
- **Animations and Transitions**
- **Multiple Backgrounds**
- **Custom Fonts with @font-face**
- **Rounded Corners (border-radius)**
- **Shadows (box-shadow, text-shadow)**

3. CSS Syntax

```
selector {
  property: value;
}
```

Example:

```
p {
  color: blue;
  font-size: 16px;
}
```

4. CSS Style Properties
Text Properties

```
color: red;
font-family: Arial, sans-serif;
font-size: 18px;
text-align: center;
text-decoration: underline;
```

Box Properties

```
width: 200px;
height: 100px;
padding: 10px;
margin: 20px;
border: 1px solid black;
```

Background Properties

```
background-color: lightblue;
background-image: url("bg.jpg");
background-repeat: no-repeat;
background-size: cover;
```

Block/Layout Properties

```
display: block | inline | flex | grid;
float: left | right;
clear: both;
```

overflow: hidden;

Positioning Properties

position: static | relative | absolute | fixed | sticky;
top: 10px; left: 20px;
z-index: 5;

List and Table Properties

list-style-type: square;
list-style-position: inside;

border-collapse: collapse;
table-layout: fixed;

Cursor Property

cursor: pointer; /* or move, text, wait, crosshair */

5. CSS Selectors

Selector Type	Syntax	Description
Tag	h1 {}	Selects all <h1> elements
ID	#title {}	Selects element with id="title"
Class	.note {}	Selects all elements with class="note"
Group	h1, p {}	Selects all <h1> and <p>
Child	div > p {}	Selects <p> elements that are direct children of
Descendant	div p {}	Selects all <p> inside a
Adjacent Sibling	h1 + p {}	Selects the <p> immediately after <h1>
General Sibling	h1 ~ p {}	Selects all <p> after <h1> in the same parent
Attribute	input[type="text"] {}	Selects input elements with type="text"
First Line	p::first-line {}	Styles the first line of text in a paragraph
Before/After	p::before { content: "*"; }	Inserts content before/after an element
Sub-class	ul li {}	Targets inside
Pseudo-class	a:hover {}	Selects when user hovers over a link

Example: Complete CSS3 Styling

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      background-color: #f0f0f0;
      font-family: Verdana;
    }

    h1 {
      color: navy;
      text-align: center;
    }
```

```
    .box {
      width: 300px;
      height: 200px;
      border: 2px solid black;
      background-color: lightblue;
      margin: 20px auto;
      padding: 10px;
    }

    #special {
      color: red;
      font-weight: bold;
    }

    p::first-line {
      font-style: italic;
    }

    a:hover {
      color: green;
    }
  </style>
</head>
<body>
  <h1>Welcome</h1>
  <div class="box">
    <p id="special">This is a styled box with CSS3.</p>
    <a href="#">Hover over me</a>
  </div>
</body>
</html>
```

UNIT 4: JavaScript / DOM / AJAX

JavaScript Basics

- Lightweight, interpreted language for scripting web pages.
- Runs in the browser or on servers (Node.js).

Variables & Data Types

- Declaration:
  - `var` (legacy), `let`, `const`
- Data Types:
  - Primitive: string, number, boolean, null, undefined
  - Composite: object, array, function

Statements, Operators, and Functions

- **Control Statements:** `if`, `switch`, `for`, `while`
- **Operators:** `+`, `-`, `*`, `/`, `=`, `==`, `===`, `&&`, `||`, `!`
- **Function Declaration:**

```
function greet(name) {
  return "Hello " + name;
}
```

JavaScript Objects

- Store data as key-value pairs
- Example:

```
let person = {
  name: "Alice",
  age: 25,
  greet: function() { return "Hi " + this.name; }
};
```

Arrays and Built-in Objects

```
let fruits = ["apple", "banana", "cherry"];
console.log(fruits.length); // 3
```

- Useful Objects: `Date`, `Math`, `RegExp`, `String`, `Array`

Debugging

- Use `console.log()` to inspect values
- Browser DevTools for breakpoints and step-through debugging

Host Objects

- Provided by the browser environment

- Common examples:
    - `window`, `document`, `navigator`, `location`

DOM (Document Object Model)

- Represents page structure as a tree of objects
- DOM Access:

```
document.getElementById("title");
document.querySelector(".menu-item");
```

- Modifying content:

```
document.getElementById("title").textContent = "New Title";
```

DOM Event Handling

- Detect and handle user actions:

```
document.getElementById("submit").addEventListener("click", function() {
  alert("Form Submitted!");
});
```

AJAX (Asynchronous JavaScript and XML)

- Allows web pages to load data in the background without refreshing
- Example:

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "data.json", true);
xhr.onload = function() {
  if (xhr.status === 200) {
    let data = JSON.parse(xhr.responseText);
    console.log(data);
  }
};
xhr.send();
```

- Modern alternative using `fetch()` API:

```
fetch("data.json")
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error("Error:", error));
```

1. JavaScript: Introduction

What is JavaScript?

- JavaScript is a **client-side scripting language**.
- It runs in the browser and can interact with HTML and CSS.

- Used for **form validation, DOM manipulation, animations, AJAX** calls, etc.

2. JavaScript Basics

Syntax Example

```
console.log("Hello, JavaScript!");
```

Variables and Data Types

```
// Variable declaration
let name = "John";    // String
const age = 30;      // Number
var isStudent = true; // Boolean

// Data types: string, number, boolean, null, undefined, object, symbol
```

Statements & Operators

```
if (age > 18) {
  console.log("Adult");
}

let sum = 10 + 5; // Arithmetic
let isValid = true && false; // Logical
```

Literals

- **Number**: 10, 3.14
- **String**: "Hello"
- **Boolean**: true, false
- **Array/Object**: [1, 2, 3], { name: "Alice" }

3. Functions in JavaScript

Example:

```
function greet(name) {
  return "Hello, " + name;
```

```
}
console.log(greet("Alice"));
```

4. JavaScript Objects

Properties, References, and Methods

```
let person = {
  name: "John",
  age: 25,
  greet: function() {
    return "Hi, I'm " + this.name;
  }
};
console.log(person.greet());
```

Arrays

```
let colors = ["red", "green", "blue"];
colors.push("yellow"); // Add to array
```

Built-in Objects

- Math, Date, Array, String, Object

```
let today = new Date();
console.log(Math.sqrt(16)); // 4
```

Debugging

- Use console.log(), debugger;, or browser DevTools.

5. Host Objects

- **Host objects** are provided by the browser, not by JavaScript.
- Examples: window, document, location, navigator, console

6. Document Object Model (DOM)

What is the DOM?

- DOM = Document Object Model
- Represents the structure of the HTML document as a tree

Document Tree

```
<html>
 <body>
  <div>
   <p>Hello</p>
  </div>
 </body>
</html>
```

Accessing DOM Elements

```
document.getElementById("myId");
document.querySelector("p");
```

DOM Manipulation Example

```
<p id="demo">Hello</p>
<script>
 document.getElementById("demo").innerText = "Welcome!";
</script>
```

7. DOM Event Handling

Inline Event

```
<button onclick="alert('Clicked')">Click Me</button>
```

JS Event Listener

```
document.getElementById("btn").addEventListener("click", function() {
```

```
  alert("Button Clicked");
});
```

8. Basic Introduction to AJAX

What is AJAX?

- **AJAX** = Asynchronous JavaScript and XML
- Allows sending/receiving data **without reloading** the page.

AJAX with fetch()

```
fetch("https://api.example.com/data")
  .then(response => response.json())
  .then(data => console.log(data));
```

Real-World Use:

- Submitting forms without reloading
- Loading data from a server
- Chat applications, dynamic content updates