**What Are Web Technologies?**

> Tools, languages, and systems used to build and run websites/web apps

> Like ingredients and tools for running an online “lemonade stand”

Examples: HTML, CSS, JavaScript, servers, APIs

Important because they make the internet possible

Main Components:

Frontend – What the user sees and interacts with (HTML, CSS, JavaScript)

→ Like the sign, cups, and cashier

Backend – Behind-the-scenes processing and data storage (Python, PHP, databases)

→ Like the kitchen in a lemonade stand

Data Formats – Standard formats for exchanging data (JSON, XML)

→ Like universally understood order forms

APIs – Connect different parts/systems

→ Like a worker carrying orders between customer and kitchen

Web Protocols – Rules for sending data (HTTPS, DNS)

→ Like delivery rules and traffic signs

Graphics – Images, animations, videos

→ Like decorations at the stand

How They Work Together

> User opens a browser and visits a website

> Protocols connect to a server

> Frontend loads (HTML, CSS, JS)

> User interacts (e.g., presses a button)

> API sends request to backend

> Backend gets data (database) and sends it back (JSON)

Graphics make everything visually appealing

→ Analogy: A customer orders lemonade and it’s prepared in the kitchen and served

State of Web Technologies in 2025

> Responsive design using Flexbox & Grid

> Highly interactive with React, Vue, etc.

> APIs + cloud services for real-time data

> HTTPS is standard for security

> WebGL/3D graphics for immersive experiences

> Accessibility and semantic HTML for all users

Why Learn Web Technologies

> Build anything (blogs, shops, games)

> Strong career opportunities

> Encourages creativity

> Foundation for advanced topics like AI and backend

> Used by every modern organization

The Web Technology Ecosystem (Diagram Highlights)

> User (Browser): visits a site

> Frontend: displays content

> API: sends/receives data (JSON)

> Backend: handles logic

> Server/Database: stores info

> Graphics: adds visual content

> HTTPS: secures the communication

Getting Started

> Install a text editor (e.g., VS Code)

> Try a simple HTML/CSS page

> Use tools like CodePen or Replit

> Learn frontend first (HTML/CSS → JavaScript → backend)

> Use online resources (W3Schools, freeCodeCamp)

Tips for Beginners

> Start simple and build gradually

> Experiment and refresh your browser

> Ask questions and look up solutions

> Practice daily, even for a few minutes

> Make fun projects about your interests!

**What Are Frontend Languages?**

> Tools that build the parts of a website users see and interact with.

> Run in the browser (Chrome, Firefox, etc.).

> Analogy: The sign, cups, and worker at a lemonade stand — everything the customer sees.

The Big Three:

– HTML = structure

– CSS = style

– JavaScript = interactivity

Why it matters: Makes websites attractive and user-friendly.

HTML – Structure

> Uses tags (e.g., <h1>, <p>) to organize content.

> Like the frame or tables of a lemonade stand.

> Accessible and readable to both browsers and screen readers.

>First layer of every webpage.

CSS – Style

> Styles HTML elements (colors, sizes, layout).

> Adds visual appeal and responsiveness.

> Supports animations and responsive design.

> Turns simple content into professional-looking pages.

JavaScript – Interactivity

> Adds behavior to pages (buttons, pop-ups, sliders).

> Runs in the browser and responds to user actions.

> Can fetch data (APIs) and work with frameworks like React.

> Makes pages feel like applications.

How They Work Together

> HTML loads and builds the structure.

> CSS styles the structure.

> JavaScript adds actions and interactions.

> Together they create a complete, modern webpage.

Frontend Languages in 2025

> HTML5: Supports video, canvas, and improved accessibility.

> CSS3: Advanced layouts (flexbox/grid), animations, responsive design.

> JavaScript: Frameworks (React, Vue, Svelte) power rich, app-like sites.

> Trends: Responsiveness, interactivity, and accessibility.

Diagram Concept

> HTML → Structure

> CSS → Style

> JavaScript → Interactivity

> Finished Webpage → Styled and interactive in the browser

Getting Started

1. Install a text editor (e.g., VS Code).

2. Create a simple HTML file.

3. Add CSS and JavaScript.

4. Open in a browser and test.

5. Experiment and learn incrementally (HTML → CSS → JS).

Tips for Beginners

> Start small and build up.

> Practice daily and experiment.

> Use online tools like CodePen or Replit.

> Follow tutorials (W3Schools, freeCodeCamp).

Why Learn Frontend Languages

> Build cool websites and projects.

> In-demand career skill.

> Offers creative freedom.

> Foundation for learning frameworks and advanced development.

**What is HTML**

> Stands for HyperText Markup Language

> Used to create and structure content on webpages

> Analogy: Like a skeleton that provides structure to the human body

Purpose of HTML

> Structures content (text, headings, images, etc.)

> Creates hyperlinks to connect different pages

> Supports embedding of media (images, videos, audio)

> Works together with CSS (styling) and JavaScript (interactivity)

> Universal standard and foundation of almost all websites

History of HTML

> Created in 1990 by Tim Berners-Lee (inventor of the World Wide Web)

> HTML 1.0 (1993): Basic text and links

> HTML 2.0 (1995): Added forms and images; standardized by IETF

> HTML 3.2 (1997): Introduced tables and more styling options

> HTML 4.01 (1997): Included official support for CSS and JavaScript

> HTML 5 (2014): Added modern features like video, audio, and canvas; improved interactivity and mobile support

How HTML Works

> Uses “tags” (e.g., <p>) to structure webpage content

> Tags are instructions enclosed in angle brackets

> Most tags have an opening <tag> and a closing </tag>

**What is CSS**

> Stands for Cascading Style Sheets

> Styles HTML elements (colors, fonts, layout, etc.)

> Analogy: An interior designer decorating a plain house (HTML)

> “Cascading” means more specific rules override general ones

> Example: h1 { color: red; }

Purpose of CSS

> Style elements (text, images, buttons)

> Create page layouts (grids, menus)

> Improve user experience and readability

> Keep styling consistent across pages

> Make pages responsive on different devices

History of CSS

> 1996 (CSS1): Basic styling (colors, fonts)

> 1998 (CSS2): Layouts, positioning, hover

> 2011–now (CSS3): Animations, shadows, responsive design, grid and flexbox

> CSS now supports advanced visual features used in all modern websites

How CSS Works

> Uses rules with:

> Selector (targets element)

> Declaration (style to apply)

3 ways to add CSS:

> Internal (<style> tag)

> External (.css file + <link>)

> Inline (style="" in tag — not recommended)

> Specificity decides which rule is applied if there’s a conflict

Key CSS Concepts

> Selectors: tag (p), class (.warning), id (#title)

> Box Model: content → padding → border → margin

> Display Types: block, inline, flex, grid

> Responsive Design: uses media queries (e.g., @media (max-width: 600px))

Common CSS Properties

> Colors: color, background-color

> Fonts: font-family, font-size

> Spacing: padding, margin

> Borders: border

> Layout: display, width, height

> Animation: animation, @keyframes

How to Use CSS (Workflow)

> Create a basic HTML file

> Add CSS (internal or external preferred)

> Save and test/refresh in browser

> Example: change body background with

body { background-color: lightblue; }

Diagram Concept

> HTML = structure

> CSS = styles

> Browser = renders final styled page

Practical Example Highlights

> Styled page with colored background, centered headings, and styled paragraph/button

Demonstrates combining HTML and CSS for a clean layout

Tips for Beginners

> Style one element at a time

> Experiment and refresh browser often

> Use VS Code or CodePen

> Learn from examples and inspect pages

> Practice on your own webpage

Why Learn CSS

> Makes websites visually appealing

> Valuable skill for web careers

> Enables creativity in design

> Prepares you for JavaScript and advanced styling

**Introduction to JavaScript**

> Programming language that adds interactivity to web pages

> Runs in the browser to handle actions (clicks, updates, data fetching)

> Makes websites feel like usable apps

JavaScript in HTML

> Added using <script> tag (internal, external, or inline)

> Internal: code inside <script> tags

> External: separate .js file + <script src="">

> Inline: added directly in HTML element (e.g., onclick)

> Example given for showing an alert when clicking a button

Basic JavaScript Objects

> Objects = collections of data and functions

> Common objects:

< window (browser window)

< document (webpage)

< Array (list of items)

< Object literal (custom objects with properties)

> Example: window.alert(), arrays, and object with properties

JavaScript Operators

> Operators perform actions on data

< Arithmetic: +, -, \*, /

< Assignment: =

< Comparison: ==, >, <

< Logical: &&, ||

> Example of checking if a user has enough money

Data Types

> Different types of data used in JavaScript:

< Number, String, Boolean

< Array, Object

< null, undefined

> Example combines several data types in a small script

Document Object Model (DOM)

> JavaScript uses the DOM to interact with HTML

> Allows accessing and changing HTML elements

> Key methods:

< getElementById(), querySelector()

< .innerHTML, .style

> Example updates text and color of a paragraph

Debugging

> Process of finding and fixing coding errors

> Common error types: syntax, reference, logic

> Tools: console.log, developer tools, alert()

> Example demonstrating a logic fix using Number()

Diagram Explanation (JS & DOM interaction)

> HTML = structure

> DOM = model used by JavaScript

> JavaScript manipulates DOM → interactive webpage

> Browser renders updated content

Practical Example

> Complete HTML + CSS + JavaScript page (Lemonade Stand)

> Demonstrates:

< DOM manipulation

< Data types

< Operators

< Console logging for debugging

Tips for Beginners

> Start with small scripts

> Use browser console to test code

> Try changing values and experimenting

> Use tools like VS Code/CodePen

> Try tutorials (W3Schools, freeCodeCamp)

> Practice debugging with console.log

Why Learn JavaScript

> Adds interactivity to websites

> Valuable career skill

> Encourages creativity and dynamic apps

> Next step → explore frameworks like React or Vue