UNIT-I:

HTML- Basic HML, the document body, Text, Hyperlinks, Adding More Formatting, Lists, Using Color and Images, Images, Tables, Frames, Forms-Toward Interactivity. Cascading Style sheets - Introduction, Inline Styles, Embedded Style Sheets, Linking external sheets, Backgrounds, text flow and box model.

UNIT-II:

JavaScript- Introduction, simple programming, Obtaining User Input with prompt Dialogs, Operators (arithmetic, Decision making, assignment, logical, increment and decrement). Control Structures - if… else selection statement, while, do… while repetitions statement, for statement, switch statement, break and continue statements.

UNIT-III:

Functions - program modules in JavaScript, programmer defined functions, function definition, Random-number generator, scope rules, global functions, recursion. JavaScript: Arrays, JavaScript: Objects - Math Object, String Object, Date Object, Boolean & Number Object, document and window Objects.

UNIT-IV:

Event Model - on click, on load, onerror, onmouseover, onmouseout, onfocus, onblur, onsubmit, onreset, more DHTML events. XML - Introduction, XML Basics, Structuring Data, XML Namespaces, Document Type Definitions (DTDs), W3C XML Schema Documents, XML Vocabularies, MathML, Other Markup Languages, Extensible Style sheet Language and XSL Transformations, Document Object Model (DOM).

1. INTRODUCTION (HTML)

In the modern digital world, most information is accessed through web pages, and HTML (HyperText Markup Language) forms the fundamental building block of these pages. It is a standard language used to describe the structure and organization of content on the World Wide Web.

*HTML is called a markup language because it uses special symbols called **tags** to "mark up" different parts of a document. These tags do not perform calculations or logical operations like programming languages; instead, they define how content is arranged and displayed in a web browser. For example, HTML can specify headings, paragraphs, images, links, and tables, giving a clear structure to the information.*

*The term **HyperText** refers to the concept of linking one document to another using hyperlinks. This feature allows users to move easily between different web pages, creating a connected network of information. The term **Markup Language** indicates that HTML provides instructions about how content should be structured rather than how it should behave.*

An HTML document follows a hierarchical structure, where elements are nested within one another. The main sections of a document include the <head>, which contains metadata such as the title of the page, and the <body>, which contains the visible content. This structured approach helps browsers interpret and render the content correctly.

*HTML alone is responsible only for the structure of a web page. To enhance appearance and functionality, it is commonly used along with **CSS (Cascading Style Sheets)** for styling and **JavaScript** for adding interactivity. Together, these technologies form the core of web development.*

One of the key strengths of HTML is its simplicity and universality. It can be written using a simple text editor and is supported by all web browsers, making it accessible to beginners while still being powerful enough for building complex web applications when combined with other technologies.

In summary, HTML serves as the backbone of the web by providing a standardized way to structure and present content, enabling effective communication and navigation across the internet.

1.1 Introduction to Web Technology

Web technology refers to the collection of tools, standards, protocols, and software used to create, access, and manage information on the World Wide Web. It provides the foundation for

developing websites and web applications that can be accessed through web browsers over the internet.

At its core, web technology enables communication between users and servers. When a user requests a web page through a browser, the request is sent to a web server, which processes it and returns the required content. This interaction is made possible through protocols such as HTTP (HyperText Transfer Protocol), which governs how data is transmitted over the web.

*Web technology is broadly classified into **client-side** and **server-side** technologies. Client-side technologies, such as HTML, CSS, and JavaScript, are executed in the user's browser and are responsible for the structure, design, and interactivity of web pages. Server-side technologies, such as PHP, JSP, and databases, operate on the web server and handle data processing, storage, and business logic.*

Another important aspect of web technology is its layered architecture. Each layer has a specific role:

- *The **presentation layer** handles the user interface and display of content.*
- *The **application layer** processes user requests and controls the logic.*
- *The **data layer** manages storage and retrieval of data.*

Web technology has evolved significantly over time, moving from simple static web pages to highly interactive and dynamic applications. Today, it supports features such as multimedia integration, real-time communication, e-commerce systems, and cloud-based services.

One of the key advantages of web technology is its platform independence. Web applications can run on different devices and operating systems as long as a web browser is available. This makes it a universal medium for information sharing and communication.

In conclusion, web technology plays a vital role in modern computing by enabling the creation of interactive, accessible, and scalable web applications, thereby connecting people and information across the globe.

1.1.1 Art of Creating a Website

Introduction

The art of creating a website is a systematic and creative process of designing, structuring, and developing web pages to present information effectively on the internet. It is called an "art" because it requires a thoughtful combination of technical skills, visual creativity, and understanding of user behavior. A well-designed website not only functions correctly but also provides a meaningful and engaging experience to users.

Purpose and Goal Identification

Every website is created with a specific objective, such as providing information, offering services, conducting business, or entertainment. Identifying the purpose helps in deciding:

- *What content should be included*
- *How it should be organized*
- *What features are required*

A clearly defined goal ensures that the website remains focused and effective.

Understanding the Target Audience

A successful website is always designed with the end user in mind. Understanding the target audience involves analyzing their needs, preferences, and expectations.
This helps in:

- *Choosing suitable language and content*
- *Designing user-friendly interfaces*
- *Improving accessibility and usability*

Content Organization (Information Structure)

Content is the core of any website. It must be organized in a logical and structured manner.
This includes:

- *Dividing content into sections and subsections*
- *Using headings, paragraphs, and lists*
- *Maintaining a clear hierarchy of information*

Proper organization helps users easily understand and navigate the website.

Page Structure and Semantic Design

A website is built using structured elements such as header, body, sections, and footer. Semantic design means using appropriate elements to describe the meaning of content.
Advantages:

- *Improves readability*
- *Enhances accessibility*
- *Helps search engines understand content*

Layout and Visual Design

Layout refers to the arrangement of elements on a web page. A good layout ensures:

- *Proper alignment of content*
- *Balanced distribution of elements*
- *Clear separation between sections*

Visual design includes colors, fonts, images, and spacing. These elements should be consistent and attractive to improve user engagement.

Navigation System

Navigation is the method by which users move between different pages or sections of a website.

A good navigation system should be:

- *Simple and easy to understand*
- *Consistent across all pages*
- *Quick to access important information*

Common navigation tools include menus, links, and buttons.

User Experience (UX) and Usability

User experience focuses on how easily and efficiently users can interact with the website.
Important aspects:

- *Easy-to-use interface*
- *Fast loading speed*
- *Mobile compatibility*
- *Clear instructions*

Good usability ensures user satisfaction and encourages repeated visits.

Interactivity and Functionality

Modern websites provide interactive features to enhance user engagement.
Examples:

- *Forms for user input*
- *Buttons and dynamic content*
- *Real-time updates*

These features are implemented using technologies like JavaScript and server-side scripting.

Performance and Optimization

Website performance is crucial for user retention.
Key factors:

- *Fast loading pages*
- *Optimized images and content*
- *Efficient coding practices*

A slow website may lead to poor user experience.

Testing and Maintenance

After development, the website must be tested for:

- *Errors and bugs*
- *Browser compatibility*
- *Responsiveness*

Regular updates and maintenance ensure that the website remains secure, accurate, and up to date.

1.1.2 Hypertext

Hypertext is an advanced method of organizing, linking, and accessing information in a non-sequential manner. It represents a shift from traditional linear documentation to a dynamic, interconnected system where users can navigate content based on their interests. This concept is fundamental to the structure and functioning of the World Wide Web.

The idea of hypertext originated from the need to manage large amounts of information efficiently. Traditional documents present information in a fixed order, which limits flexibility. Hypertext overcomes this limitation by allowing information to be broken into smaller units and interconnected through links.

hypertext is based on two main components:

- ***Nodes**: Individual units of information (documents, sections, or pages)*
- ***Links**: Connections that allow navigation between nodes*

This model creates a network of information rather than a single linear path.

Non-Linear Information Architecture

*One of the most important characteristics of hypertext is its **non-linear structure**. In contrast to books or printed materials, hypertext allows multiple pathways through content.*
This means:

- *Users are not restricted to a predefined sequence*
- *Information can be accessed in different orders*
- *Learning and exploration become user-driven*

This flexibility enhances understanding and allows personalized navigation.

Hyperlinks: The Core Mechanism

Hyperlinks are the operational elements of hypertext. They act as bridges connecting different nodes within the same document or across multiple documents.
Types of hyperlinks include:

- ***Internal links** (within the same page or site)*
- ***External links** (to other websites)*
- ***Anchored links** (to specific sections within a page)*

When activated, a hyperlink sends a request to retrieve the linked resource, enabling seamless navigation.

Cognitive and Usability Perspective

From a cognitive point of view, hypertext mirrors the way humans think—associatively rather than linearly. It allows users to follow connections between ideas, improving comprehension and engagement.
However, excessive linking can lead to ***cognitive overload****, where users may feel lost or confused. Therefore, proper design and structured linking are essential for effective usability.*

Role in Web Technology

Hypertext forms the backbone of web technology. HTML (HyperText Markup Language) is specifically designed to implement hypertext through anchor tags.
Its role includes:

- *Connecting web pages globally*
- *Enabling navigation across websites*
- *Integrating multimedia content*
- *Supporting interactive user experiences*

Without hypertext, the web would function as isolated pages rather than an interconnected system.

Hypertext vs Hypermedia

Hypertext is often extended to ***hypermedia****, which includes not only text but also images, audio, video, and animations.*

- ***Hypertext*** *→ Text-based linking*
- ***Hypermedia*** *→ Multimedia-based linking*

Modern web applications largely operate on hypermedia principles.
Advantages of Hypertext

- *Provides flexible and user-controlled navigation*
- *Enables quick access to related information*
- *Enhances learning through interconnected content*
- *Supports large-scale information systems*

1.1.3 Web Design Principles

Simplicity

Simplicity is a fundamental principle of web design that focuses on presenting content in a clear and straightforward manner by eliminating unnecessary elements. A simple design avoids clutter, reduces distractions, and helps users concentrate on the main information. By using

minimal text, limited graphics, and a clean layout, simplicity improves usability and enhances the overall effectiveness of a website.

Consistency

Consistency refers to maintaining uniformity in design elements such as colors, fonts, layouts, and navigation across all pages of a website. A consistent design helps users become familiar with the interface, reducing confusion and improving ease of use. It also strengthens the visual identity of the website and ensures a smooth user experience.

Visual Hierarchy

Visual hierarchy is the arrangement of elements in a way that highlights their importance. By using variations in size, color, contrast, and positioning, designers can guide users' attention to key information first. A well-defined hierarchy allows users to quickly scan and understand the content structure without difficulty.

Balance and Alignment

Balance refers to the proper distribution of visual elements on a web page to create stability and harmony. Alignment ensures that elements are placed in an organized manner, creating a neat and structured layout. Together, balance and alignment enhance readability and give the website a professional appearance.

Typography and Readability

Typography involves the selection and arrangement of text to ensure clarity and readability. Using appropriate font styles, sizes, and spacing makes content easier to read and understand. Proper typography also helps in distinguishing headings from body text, thereby improving the overall presentation of information.

Color and Contrast

Color and contrast play an important role in enhancing the visual appeal and usability of a website. Appropriate color combinations can attract attention and create a pleasant user experience, while sufficient contrast between text and background ensures readability. Poor color choices can make content difficult to view and reduce effectiveness.

Navigation

Navigation refers to the system that allows users to move through different sections of a website. A well-designed navigation structure should be simple, clear, and consistent, enabling users to find information quickly. Effective navigation reduces user effort and improves overall usability.

User Experience (UX)

User experience focuses on how users interact with a website and how easily they can achieve their goals. A good user experience ensures that the website is intuitive, efficient, and satisfying to use. Factors such as ease of use, clarity, and responsiveness contribute to a positive user experience.

Responsiveness

Responsiveness ensures that a website adapts to different screen sizes and devices such as mobile phones, tablets, and desktops. A responsive design automatically adjusts layout and content to provide a consistent experience across all platforms, increasing accessibility and usability.

Accessibility

Accessibility ensures that a website can be used by all users, including those with disabilities. This involves providing features such as alternative text for images, clear navigation, and compatibility with assistive technologies. Accessible design promotes inclusivity and wider reach.

Performance and Loading Speed

Performance refers to how quickly a website loads and responds to user actions. Fast-loading websites improve user satisfaction and reduce the chances of users leaving the site. Optimization techniques such as reducing file sizes and efficient coding help improve performance.

Interactivity and Feedback

Interactivity involves enabling users to engage with the website through elements such as buttons, forms, and animations. Feedback provides responses to user actions, such as confirmations or alerts. These features make the website more dynamic and improve user engagement.

1.2 Markup Language (HTML) – Introduction

A markup language is a system used to annotate and structure a document by embedding special symbols or tags within the content. These tags define the arrangement, organization, and meaning of the data without affecting the actual content itself. In the context of web technology, HTML (HyperText Markup Language) is the standard markup language used to create and design web pages.

HTML provides a structured way of presenting information on the World Wide Web. It uses predefined tags to describe different parts of a web page, such as headings, paragraphs, images,

links, and tables. These tags help web browsers understand how to display the content to users. Unlike programming languages, HTML does not perform computations or decision-making processes; instead, it focuses on defining the layout and structure of web content.

The term HyperText refers to the ability of HTML to link documents through hyperlinks, allowing users to navigate between web pages easily. The term Markup Language indicates that HTML uses tags to "mark up" content, specifying how it should be organized and displayed.

HTML acts as the foundation of web development and works together with other technologies such as CSS for styling and JavaScript for adding interactivity. Its simplicity, flexibility, and universal support make it an essential tool for creating web pages.

In summary, HTML is a fundamental markup language that enables the structured presentation of content on the web, forming the backbone of all modern websites.

1.2.1 HTML Structure

Introduction

HTML structure refers to the formal and logical organization of elements within an HTML document that determines how content is defined, arranged, and interpreted by web browsers. It provides a standardized framework that separates different types of information and ensures that web pages are rendered in a consistent and meaningful manner. A well-defined HTML structure is essential for clarity, accessibility, and efficient processing of web content.

Nature of HTML Document Structure

An HTML document is inherently structured in a hierarchical manner, where elements are arranged in levels forming a tree-like model. Each element represents a specific component of the document and contributes to the overall organization of content. This hierarchical structure establishes relationships between elements and enables systematic interpretation by browsers.

Example of HTML Structure

```
<!DOCTYPE html>  // Document Type Declaration

<html>  // Root Element

<head>  // Head Section

    <title>HTML Structure Example</title>  // Title of Page

    <meta charset="UTF-8">  // Metadata
```

```
</head>
<body>  // Body Section
   // Main Heading
   <h1>Welcome to My Website</h1>
   // Sub Heading
   <h2>About This Page</h2>
   // Paragraph
   <p>This page explains the basic structure of HTML.</p>
   // Another Sub Heading
   <h3>Visit Link</h3>
   // Hyperlink
   <a href="https://example.com">Click Here</a>
</body>
</html>
```

DOCTYPE Declaration and Standardization

The <!DOCTYPE html> declaration is the initial component of an HTML document. It defines the document type and specifies that the content follows HTML5 standards. This declaration ensures that browsers render the document in standards-compliant mode, thereby avoiding inconsistencies and ensuring uniform behavior across different platforms.

Root Element and Document Scope (<html>)

The <html> element serves as the root element of the document and encloses all other elements. It defines the scope and boundary of the HTML document. All structural and content elements are nested within this root, making it the fundamental container that establishes the document framework.

Head Section and Metadata Management (<head>)

The <head> section contains metadata, which is information about the document rather than its visible content. This section plays a critical role in defining how the document is interpreted, processed, and indexed.

It typically includes elements such as:

- *<title> for specifying the document title*
- *<meta> for defining character encoding, viewport settings, and other metadata*
- *<link> for connecting external resources such as stylesheets*
- *<script> for including scripts*

Although the head section is not directly visible to users, it significantly influences the behavior, accessibility, and performance of the web page.

Body Section and Content Representation (<body>)

The <body> section contains the actual content of the web page that is displayed to users. It represents the visible interface and includes elements such as text, headings, images, links, lists, tables, and forms.

The body section is responsible for presenting information in a structured and readable manner, enabling user interaction and communication of content.

Hierarchy, Nesting, and Parent–Child Relationships

HTML structure is based on a nested arrangement of elements, where elements are placed inside other elements to form parent–child relationships. This hierarchical organization ensures logical grouping of content and defines how elements relate to one another.

Proper nesting is essential for:

- *Accurate interpretation by browsers*
- *Maintaining logical consistency*
- *Preventing structural errors*

This hierarchical model is fundamental to understanding document organization.

Document Object Model (DOM) Perspective

The structured arrangement of HTML elements is represented internally by the browser as the Document Object Model (DOM). The DOM is a tree-like representation where each element is treated as a node.

This model enables:

- *Dynamic manipulation of content*
- *Interaction through scripts*
- *Real-time updates to the structure and presentation*

Thus, HTML structure serves as the basis for DOM construction.

Separation of Structure, Presentation, and Behavior

HTML structure follows the principle of separation of concerns. It focuses only on defining the structure and meaning of content, while other aspects are handled separately:

- *HTML defines structure*
- *CSS controls presentation and layout*
- *JavaScript manages behavior and interactivity*

This separation improves maintainability, scalability, and clarity of web development.

Role in Accessibility and Search Optimization

A properly structured HTML document enhances accessibility by enabling assistive technologies to interpret content effectively. It also improves search engine optimization (SEO) by providing meaningful structure that search engines can analyze and index.

Importance of Standard HTML Structure

Maintaining a standard HTML structure is essential because it:

- *Ensures consistent rendering across browsers*
- *Improves readability and maintainability of code*
- *Supports integration with other web technologies*
- *Enhances user experience and accessibility*
- *Provides a strong foundation for advanced web development*

1.2.2 Formatting Text

Formatting text in HTML refers to the process of enhancing the appearance and presentation of textual content on a web page. It involves the use of specific HTML elements to modify the style, emphasis, and structure of text so that it becomes more readable and visually appealing. Text formatting plays an important role in improving user understanding and highlighting important information.

Purpose of Text Formatting

Text formatting is used to:

- *Improve readability of content*
- *Highlight important information*
- *Organize content clearly*
- *Enhance visual presentation*

Proper formatting helps users quickly identify key points and understand the structure of the content.

Basic Text Formatting Elements

Basic text formatting elements in HTML are used to modify the appearance of text to improve readability and presentation. These elements are mainly concerned with the visual styling of content and do not add semantic meaning. They help in emphasizing important parts of text and organizing information clearly on a web page.

Table: Basic Text Formatting Elements

Element	***Concept / Definition***	***Syntax***	***Example Code***	***Output***
**	*Used to display text in bold form to draw attention to important content. It is a physical formatting tag.*	*text*	*Important*	***Important***
**	*Used to display text in italic style, often for emphasis or special terms. It changes the appearance only.*	*text*	*Example*	*Example*
<u>	*Used to underline text, highlighting specific parts of content. It affects only the visual presentation.*	*<u>text</u>*	*<u>Text</u>*	*<u>Text</u>*
* *	*Inserts a line break within text without creating a new paragraph. It controls text flow.*	* *	*Line1 Line2*	*Line1* *Line2*
<hr>	*Creates a horizontal line to separate sections of content, improving structure and clarity.*	*<hr>*	*<hr>*	———————

xample

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>Basic Formatting</h2>

<p><b>Bold Text</b></p>
<p><i>Italic Text</i></p>
<p><u>Underlined Text</u></p>

<p>Line One<br>Line Two</p>

<hr>
</body>
</html>
```

Output Explanation

- *Bold text appears darker and thicker*
- *Italic text appears slanted*
- *Underlined text has a line below it*
- *Line break moves text to next line*
- *Horizontal rule creates a visible divider*

Advanced Formatting Elements

Advanced formatting elements are designed to represent specialized forms of text such as superscripts, subscripts, highlighted text, and editorial changes. These elements improve both the presentation and interpretation of content, making them important in technical and structured documents.

Table: Advanced Formatting Elements

Element	***Concept / Definition***	***Syntax***	***Example Code***	***Output***
<sup>	*Displays text slightly above the normal line, used for powers and exponents.*	*^{text}*	*X²*	*X²*
<sub>	*Displays*	*_{text}*	*H₂O*	*H_2O*

	text slightly below the normal line, used in chemical formulas.			
<mark>	Highlights text to indicate importance or reference.	<mark>text</mark>	<mark>Important</mark>	<mark>Important</mark>
<small>	Displays text in smaller font size.	text	Note	Note
<del>	Represents deleted text with a strikethrough line.	~~text~~	~~Old~~	~~Old~~
<ins>	Represents inserted text, usually underlined.	<ins>text</ins>	<ins>New</ins>	<u>New</u>

Illustrative Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Advanced Formatting</h2>

<p>X<sup>2</sup> (Superscript)</p>
```

```
<p>H<sub>2</sub>O (Subscript)</p>
<p><mark>Highlighted Text</mark></p>
<p><small>Small Text</small></p>
<p><del>Deleted Text</del></p>
<p><ins>Inserted Text</ins></p>

</body>
</html>
```

Output Explanation

- *Superscript appears above the normal line (X^2)*
- *Subscript appears below the normal line (H_2O)*
- *Mark highlights the text*
- *Small reduces font size*
- *Deleted text shows a strike line*
- *Inserted text appears underlined*

1.2.3 Commenting Code

Comments are non-executable parts of the code that do not affect the output of a web page. They are included for documentation and clarification purposes. Comments help developers explain complex sections, mark important parts of the code, and temporarily disable code during testing.

Syntax of HTML Comments

HTML comments are written using the following syntax:
`<!-- This is a comment -->`

Everything written inside <!-- and --> is treated as a comment and is not displayed in the browser.

Table: Commenting in HTML

Aspect	*Description*	*Syntax*	*Example Code*	*Output*
Comment	*Used to add notes in code without affecting output*	*<!-- comment -->*	*<!-- This is a comment -->*	*(Not displayed)*
Inline Comment	*Comment written within a line of code*	*<!-- comment -->*	* Hello <!-- Text -->*	*Hello*
Multi-line Comment	*Comment written across multiple lines*	*<!-- line1 line2 -->*	*<!-- This is multi-line comment -->*	*(Not displayed)*

1.3 HTML Elements

HTML elements are systematically classified based on their functional role in structuring, presenting, and controlling web content. This classification helps in understanding how different elements contribute to document structure, semantics, and user interaction.

Comprehensive Table of HTML Elements

Category	***Element***	***Type***	***Concept / Role***	***Syntax***	***Example***	***Output***
Document Structure	*<html>*	*Container*	*Root element defining entire document scope*	*<html>...</html>*	*<html>...</html>*	*Whole page*
	<head>	*Container*	*Holds metadata and document settings*	*<head>...</head>*	*<head>...</head>*	*Not visible*
	<title>	*Container*	*Defines page title in browser tab*	*<title>text</title>*	*<title>Page</title>*	*Tab title*
	<body>	*Container*	*Contains visible content*	*<body>...</body>*	*<body>Text</body>*	*Page content*
Text Structure	*<h1>–<h6>*	*Block*	*Defines hierarchical headings*	* text *	* Title *	*Heading*
	* *	*Block*	*Defines*	* text *	* Hello *	*Paragraph*

			paragraph content			
	 	Empty	Controls line break within text	 	A B	New line
	<hr>	Empty	Separates content visually	<hr>	<hr>	Line
Text Formatting (Physical)		Inline	Bold appearance without meaning	text	Bold	**Bold**
		Inline	Italic style	text	Italic	Italic
	<u>	Inline	Underlines text	<u>text</u>	<u>Text</u>	<u>Text</u>
Text Formatting (Semantic)		Inline	Indicates importance	text	Important	**Important**
		Inline	Indicates emphasis	text	Text	Text
	<mark>	Inline	Highlights	<mark>text</mark>	<mark>Note</mark>	<mark>Note</mark>

			content			
	<small>	Inline	Less important text	text	Note	small
	<del>	Inline	Represents removed text	~~text~~	~~Old~~	~~Old~~
	<ins>	Inline	Represents added text	<ins>text</ins>	<ins>New</ins>	<u>New</u>
	<sup>	Inline	Superscript (power)	^{text}	X²	X^2
	<sub>	Inline	Subscript (chemical)	_{text}	H₂O	H_2O
Hyperlink & Media	<a>	Inline	Creates navigation link	text	Link	Clickable
	<img>	Empty	Embeds image	<img src="">	<img src="img.jpg">	Image
	<audio>	Container	Embeds audio	<audio>...</audio>	<audio controls>	Audio
	<video>	Container	Embeds video	<video>...</video>	<video controls>	Video
Lists		Block	Unordered list	...	A	Bullets
		Block	Ordered	...	A </o	Numbers

			list		l>	
		Block	List item	text	Item	Item
Tables	<table>	Block	Defines table	<table>...</table>	<table>	Table
	<tr>	Block	Table row	<tr>...</tr>	<tr>	Row
	<td>	Block	Table data	<td>...</td>	<td>A</td>	Cell
	<th>	Block	Table heading	<th>...</th>	<th>Head</th>	Bold cell
Forms	<form>	Block	User input container	<form>...</form>	<form>	Form
	<input>	Empty	Input control	<input type="">	<input type="text">	Field
	<textarea>	Container	Multi-line input	<textarea>	<textarea>	Box
	<button>	Container	Click button	<button>text</button>	<button>OK</button>	Button
	<select>	Container	Dropdown list	<select>	<select>	Dropdown
	<option>	Container	Option item	<option>	<option>A</option>	Option
Semantic Layout	<header>	Block	Page header	<header>	<header>	Section
	<footer	Block	Page	<footer>	<footer>	Section

	>		*footer*			
	<section>	*Block*	*Section grouping*	*<section>*	*<section>*	*Section*
	<article>	*Block*	*Independent content*	*<article>*	*<article>*	*Content*
	<nav>	*Block*	*Navigation links*	*<nav>*	*<nav>*	*Menu*
	<aside>	*Block*	*Sidebar content*	*<aside>*	*<aside>*	*Side*
Layout Elements	* *	*Block*	*Generic container*	* *	* *	*Block*
	**	*Inline*	*Inline container*	**	**	*Inline*
Metadata & Script	*<meta>*	*Empty*	*Metadata info*	*<meta>*	*<meta charset="">*	*Hidden*
	<link>	*Empty*	*External resource*	*<link>*	*<link rel="">*	*Hidden*
	<style>	*Container*	*CSS rules*	*<style>*	*<style>*	*Styling*
	<script>	*Container*	*JavaScript*	*<script>*	*<script>*	*Behavior*

1.3.1 Anchors (Links)

Anchors, or links, are HTML elements used to connect one web page to another or to different sections within the same page. They enable easy navigation and form the basis of hypertext, allowing users to access related information efficiently.

Anchor Tag (<a>)

The <a> (anchor) element is used to create hyperlinks in HTML. It defines a clickable link that directs the user to another resource.

Syntax

Link Text

- *href specifies the destination of the link*
- *Link Text is the clickable content displayed to the user*

Types of Links

- ***External Links** – Connect to web pages on other websites*
- ***Internal Links** – Connect to pages within the same website*
- ***Anchor Links** – Connect to specific sections within the same page*

External Links

External links are hyperlinks that connect a web page to a resource located on a different website or domain. They use a complete URL (Uniform Resource Locator) and allow users to access information outside the current website. These links play an important role in connecting different websites and expanding access to global information.

Syntax:

Link Text

Example:

Visit Example

Internal Links

Internal links are hyperlinks that connect one page to another page within the same website. They help in organizing the website structure and enable smooth navigation between related pages. Internal linking improves user experience and helps users explore content within the site efficiently.

Syntax:

Link Text

Example:

About Us

Anchor Links

Anchor links are hyperlinks that connect to a specific section within the same web page. They use an identifier (id) to locate a particular part of the document. These links are useful for navigating long pages, allowing users to jump directly to relevant sections without scrolling.

Syntax:

Link Text

Example:

Go to Section

<h2 id="section1">Section 1</h2>

1.3.2 Images

Images are an important component of web pages used to enhance visual presentation and convey information effectively. They make web pages more attractive, engaging, and easier to understand. HTML provides a specific element to display images within a document.

In HTML, images are embedded into a web page using a dedicated element rather than being written as text. The browser fetches the image from a specified location and displays it at the desired position. Images help in improving communication by providing visual representation of content.

Image Tag (<img>)

The <img> element is used to display images in HTML. It is an empty element, meaning it does not have a closing tag.

Syntax

<img src="image.jpg" alt="description">

- *src → Specifies the path or location of the image*
- *alt → Provides alternative text if the image cannot be displayed*

Example

<img src="photo.jpg" alt="Sample Image" width="200" height="150">

This displays an image named photo.jpg with specified size.

1.3.3 Backgrounds

Backgrounds in HTML are used to enhance the visual appearance of web pages by applying colors or images behind the content. They play an important role in improving the overall design, readability, and user experience of a website.

A background refers to the area behind the content of a web page or an element. It can be customized using colors or images to make the page more attractive and visually organized. Proper use of backgrounds helps in highlighting content and maintaining visual balance.

Background Color

Background color is used to apply a solid color behind the content of a web page.

Syntax:

```
<body style="background-color: lightblue;">
```

Example:

```
<body style="background-color: lightyellow;">
   <h1>Welcome</h1>
</body>
```

👉 *This sets the background color of the page.*

Background Image

Background images are used to display an image behind the content.

Syntax:

```
<body style="background-image: url('bg.jpg');">
```

Example:

```
<body style="background-image: url('background.jpg');">
   <h1>My Page</h1>
</body>
```

👉 *This sets an image as the background.*

1.4 HTML Lists and Tables

HTML provides specialized elements for organizing and presenting information in a structured and meaningful manner. Lists and tables are two important constructs used to display grouped and tabular data respectively. While lists are used to represent related items in a sequential or non-sequential format, tables are used to present data in a grid of rows and columns. Both play a significant role in improving readability, clarity, and logical organization of web content.

1.4.1 Lists (Ordered and Unordered)

Introduction

Lists in HTML are structured elements used to represent a collection of related items in a clear and organized manner. They provide a systematic way of presenting information, enabling better readability and logical grouping of content. Lists are widely used in web pages to display menus, instructions, categories, and hierarchical data.

A list represents a set of items that share a common relationship. In HTML, lists are designed to present such items either in a specific sequence or without any defined order. This classification is based on the importance of the arrangement of items.

The use of lists enhances:

- *Logical organization of content*
- *Visual clarity and readability*
- *Ease of navigation and understanding*

Classification of HTML Lists

HTML primarily supports two types of lists based on the nature of arrangement:

1. ***Ordered Lists (Sequential Representation)***
2. ***Unordered Lists (Non-Sequential Representation)***

Each type serves a distinct purpose in content presentation.

Ordered Lists (<ol>)

An ordered list represents items in a specific sequence where the order of elements is meaningful. The items are automatically numbered or arranged in a predefined order. This type of list is used when the sequence of information affects its meaning or interpretation.

Structural Representation

The <ol> element acts as a container for ordered items, while each item is defined using the <li> element.

Syntax

```
<ol>
  <li>Item 1</li>
  <li>Item 2</li>
</ol>
```

Functional Characteristics

- *Displays items in numbered or ordered format*
- *Maintains logical sequence of information*
- *Supports different numbering styles (numbers, alphabets, Roman numerals)*

Example

```
<ol>
  <li>Login</li>
  <li>Select Course</li>
```

```
  <li>Submit Form</li>
</ol>
```

Unordered Lists (<ul>)

An unordered list represents items without any specific sequence. The order of items does not affect their meaning. Items are typically displayed using bullet points, indicating that each item is of equal importance.

Structural Representation

The <ul> element serves as the container, and <li> defines each list item.

Syntax

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

Functional Characteristics

- *Displays items using bullets*
- *Does not impose any order or priority*
- *Suitable for general listing of items*

Example

```
<ul>
  <li>HTML</li>
  <li>CSS</li>
  <li>JavaScript</li>
</ul>
```

List Item Element (<li>)

The <li> (list item) element is a fundamental component of both ordered and unordered lists. It defines each individual item within a list and acts as a container for content. The <li> element ensures uniform representation of items within the list structure.

Hierarchical Representation through Nested Lists

HTML allows lists to be nested within other lists, enabling multi-level or hierarchical representation of data. This feature is useful for representing categories, subcategories, and structured relationships.

Example

```
<ul>
  <li>Programming
    <ul>
      <li>C</li>
      <li>Java</li>
    </ul>
  </li>
</ul>
```

HTML Lists

Aspect	***Ordered List ( )***	***Unordered List ( )***
Concept	*Represents items in a specific sequence where order is important*	*Represents items where order is not important*
Nature	*Sequential (step-by-step)*	*Non-sequential (group-based)*
Semantic Meaning	*Indicates progression, ranking, or procedure*	*Indicates grouping of related items*
Display Style	*Numbers, alphabets, Roman numerals*	*Bullet points (disc, circle, square)*
Tag Used	* *	* *
Item Tag	* *	* *
Syntax	* Item *	* Item *
Example Code	* - Step 1 - Step 2 *	* - Apple - Mango *
Output	*1. Step 1* *2. Step 2*	*• Apple* *• Mango*
Use Cases	*Instructions, procedures, rankings*	*Lists of items, features, categories*
Importance of Order	*Order changes meaning*	*Order does not affect meaning*
Cognitive Role	*Helps in understanding sequence*	*Helps in grouping information*
Customization	*Can change numbering type (type, start)*	*Can change bullet style*
Nesting Support	*Supports nested lists*	*Supports nested lists*

1.4.2 Tables

Tables in HTML represent a structured mechanism for organizing data into rows and columns, forming a grid-like arrangement. They are designed to present relationships between data points

in a clear, logical, and interpretable manner. Tables transform raw data into structured information, enabling efficient comparison, analysis, and understanding.

Concept	***Explanation***
Tabular Representation	*Data is arranged in rows and columns*
Logical Organization	*Each cell represents a data unit*
Relational Structure	*Data is interconnected across rows and columns*
Grid Model	*Forms a matrix-like structure for clarity*

Core Elements of HTML Tables

Element	***Type***	***Conceptual Role***	***Syntax***	***Example***	***Output Meaning***
<table>	*Container*	*Defines the complete table structure*	*<table>...</table>*	*<table>*	*Entire table*
<tr>	*Structural*	*Represents a horizontal row*	*<tr>...</tr>*	*<tr>*	*Row*
<th>	*Semantic*	*Defines header cells (labels)*	*<th>text</th>*	*<th>Name</th>*	*Column heading*
<td>	*Data*	*Represents individual data cells*	*<td>text</td>*	*<td>Ravi</td>*	*Data value*

Table Structure Hierarchy

Example (Conceptually Complete Table)

```
<!DOCTYPE html>
<html>
<body>

<table border="1">
  <caption>Student Details</caption>

  <thead>
    <tr>
      <th>Name</th>
      <th>Marks</th>
    </tr>
  </thead>
```

```
  <tbody>
    <tr>
      <td>Ravi</td>
      <td>85</td>
    </tr>
    <tr>
      <td>Sita</td>
      <td>90</td>
    </tr>
  </tbody>

</table>

</body>
</html>
```

1.5.1 Form Elements

1.5.2 Input Controls

Input controls are interactive elements that allow users to enter data into a form. They define the type of data to be collected, such as text, numbers, or selections.

Table: Input Controls

Input Type	***Concept / Role***	***Syntax***	***Example***	***Output***
Text	*Single-line input*	*<input type="text">*	*<input type="text">*	*Text box*
Password	*Hidden text input*	*<input type="password">*	*<input type="password">*	*Password field*
Radio	*Select one option*	*<input type="radio">*	*<input type="radio">*	*Circle option*
Checkbox	*Select multiple options*	*<input type="checkbox">*	*<input type="checkbox">*	*Square option*
Submit	*Submit form*	*<input type="submit">*	*<input type="submit">*	*Button*

Reset	*Reset form data*	*<input type="reset">*	*<input type="reset">*	*Button*
Email	*Email input*	*<input type="email">*	*<input type="email">*	*Email field*
Number	*Numeric input*	*<input type="number">*	*<input type="number">*	*Number box*

Example Program

```
<!DOCTYPE html>
<html>
<body>

<form>
  <label>Name:</label>
  <input type="text"><br><br>

  <label>Password:</label>
  <input type="password"><br><br>

  <input type="submit" value="Submit">
</form>

</body>
</html>
```

1.6 Frames

Frames in HTML represent an early mechanism for dividing a browser window into multiple independent sections, each capable of displaying a separate HTML document. This approach enabled simultaneous viewing of multiple resources within a single interface, thereby introducing a modular method of presenting web content.

*The concept of frames is based on **partitioning the browser viewport** into distinct regions, where each region functions as an independent document container. Instead of loading a single continuous page, the browser renders multiple documents concurrently within predefined sections.*

This model reflects the idea of:

- ***Spatial segmentation** of content*

- ***Parallel document rendering***
- ***Independent content loading within a unified interface***

1.6.1 Introduction to Frames

In a frame-based layout, the browser window is divided using a <frameset> element rather than the <body> element. Each frame operates independently, meaning:

- *Content in one frame can change without affecting others*
- *Navigation can be fixed in one frame while content updates in another*

Core Elements of Frames

Element	***Type***	***Conceptual Role***	***Functional Description***
<frameset>	*Structural*	*Defines layout of frames*	*Divides window into rows or columns*
<frame>	*Container*	*Represents individual frame*	*Loads separate HTML document*
<noframes>	*Fallback*	*Alternative content*	*Displayed if frames not supported*

1.6.2 Types

<frameset> Element

The <frameset> element is a fundamental structural construct used to define the layout of a frame-based HTML document. It replaces the <body> element in frame-based pages and establishes the spatial arrangement of multiple frames within the browser window.

Role

*The <frameset> embodies the concept of **layout partitioning**, where the browser viewport is divided into multiple independent regions. It determines how the available display space is allocated, either horizontally (rows) or vertically (columns), thereby forming the foundational grid for frame-based rendering.*

Functional Behavior

- *It divides the browser window using rows or cols attributes*
- *It does not display content directly; instead, it acts as a container for <frame> elements*
- *It enables **parallel rendering** of multiple documents*
- *It supports nested framesets, allowing hierarchical layout design*

Structural Characteristics

Aspect	***Explanation***
Replacement Role	*Substitutes <body> in frame-based pages*
Layout Control	*Uses rows and cols attributes*

Hierarchical Nature	*Can contain nested <frameset> elements*
Rendering Logic	*Defines spatial segmentation*

Syntax

```
<frameset cols="50%,50%">
</frameset>
```

<frame> Element

The <frame> element represents an individual frame within a frameset and is responsible for loading and displaying a separate HTML document.

Role

*The <frame> acts as a **content container** that encapsulates an independent browsing context. Each frame behaves like a mini browser window, capable of loading and rendering its own document independently of other frames.*

Functional Behavior

- *Uses the src attribute to specify the document to be displayed*
- *Operates independently, allowing selective content updates*
- *Maintains isolation between different sections of the interface*
- *Supports attributes such as name, scrolling, and noresize*

Structural Characteristics

Aspect	***Explanation***
Content Loading	*Displays external HTML document*
Independence	*Operates separately from other frames*
Navigation Targeting	*Can be targeted by links*
Rendering Unit	*Acts as a self-contained display area*

Syntax

```
<frame src="page.html">
```

<noframes> Element

The <noframes> element is a fallback mechanism that provides alternative content for browsers that do not support frame-based layouts.

Role

*The <noframes> element represents the principle of **graceful degradation**, ensuring that content remains accessible even when the primary frame-based structure cannot be rendered.*

Functional Behavior

- *Displays content only when frames are unsupported*
- *Typically contains a <body> section with alternative information*
- *Enhances accessibility and compatibility*
- *Ensures continuity of user experience*

2. Dynamic Web Pages and CSS

2.1 Introduction to Dynamic Web Pages

*A **web page** is a document that is accessed through the internet using a web browser. Web pages can be broadly classified into two categories: **static web pages** and **dynamic web pages**.*

*A **dynamic web page** is a web page whose content can change automatically based on different factors such as user interaction, time, database content, or server-side processing. Unlike static pages, dynamic pages are generated at runtime, meaning the content is created when the user requests the page.*

*Dynamic web pages are widely used in modern web applications such as **online shopping websites, social media platforms, banking systems, and e-learning portals**. These pages can display personalized content, process user input, and interact with databases.*

Dynamic web pages are created using a combination of technologies, including:

- ***Client-side scripting** (e.g., JavaScript) – executes in the user's browser*
- ***Server-side scripting** (e.g., PHP, Python, Java, Node.js) – executes on the web server*
- ***Databases** (e.g., MySQL, MongoDB) – store and retrieve data dynamically*

Key Characteristics of Dynamic Web Pages:

- *Content changes based on user actions or data*
- *Interactive and responsive*
- *Can process forms and user input*
- *Connected to databases*
- *Generated in real-time*

Example:

When a user logs into a website, the page displays their name, profile details, and personalized content. This is a dynamic web page because the content changes for each user.

2.1.1 Static vs Dynamic Pages

*Web pages are broadly classified into **static** and **dynamic** based on how their content is created and delivered to the user. The following table presents a clear comparison between them:*

Basis of Comparison	**Static Web Pages**	**Dynamic Web Pages**
Meaning	*Static web pages are fixed web pages whose content does not change automatically.*	*Dynamic web pages are web pages whose content changes according to user input or data.*
Content Nature	*Content remains the same for all users.*	*Content varies from user to user.*
Creation	*Created using simple HTML and CSS.*	*Created using HTML, CSS along with scripting languages like JavaScript, PHP, etc.*
Content Generation	*Content is written once and stored on the server.*	*Content is generated at runtime when the page is requested.*
Processing	*No processing is required on the server.*	*Requires processing on the server or client side.*
Database Usage	*Does not use a database.*	*Uses databases to store and retrieve data.*
User Interaction	*Limited interaction.*	*High level of interaction is possible.*
Performance	*Faster loading due to fixed content.*	*Slightly slower due to processing.*
Complexity	*Simple to design and develop.*	*More complex to design and maintain.*
Maintenance	*Changes must be made manually in each page.*	*Content can be updated easily through database or backend.*
Examples	*Informational websites, company profiles.*	*Online shopping sites, social media platforms.*

2.1.2 Technologies for Dynamic Pages

*Dynamic web pages are built using a **set of interrelated technologies** that operate across different layers of a web system. Each technology performs a specific role in the **generation, processing, transmission, and presentation of dynamic content**.*

At a deeper level, these technologies can be organized into the following layers:

- ***Client-Side (Presentation Layer)***
- ***Server-Side (Application Layer)***
- ***Data Layer (Database Systems)***
- ***Communication & Supporting Technologies***

1. Client-Side Technologies (Presentation Layer)

*Client-side technologies are executed in the **user's web browser** and are responsible for **displaying content and enabling user interaction**. They form the **presentation layer** of a web application, which directly communicates with the user. These technologies determine how a web page looks and behaves, ensuring that it is **interactive, responsive, and user-friendly**. The main components of client-side technologies include HTML, CSS, JavaScript, and the browser's rendering engine, all of which work together to present dynamic content effectively.*

1.1 HTML (Structure Layer)

*HTML (HyperText Markup Language) provides the **basic structure** of a web page. It defines various elements such as **headings, paragraphs, images, links, and forms**, which organize the content in a meaningful way. HTML acts as the **skeleton or backbone** of a web page, upon which styling and behavior are applied. In dynamic web pages, HTML serves as the base structure that can be modified and updated dynamically using scripting technologies.*

1.2 CSS (Presentation Layer)

*CSS (Cascading Style Sheets) is used to control the **visual appearance and layout** of a web page. It defines properties such as **colors, fonts, spacing, positioning, and responsiveness**, making the page attractive and easy to use. CSS enables developers to **separate content from presentation**, which improves maintainability and consistency across multiple pages. It also supports responsive design, allowing web pages to adapt to different screen sizes and devices.*

1.3 JavaScript (Behavior Layer)

*JavaScript is a scripting language that adds **interactivity and dynamic behavior** to web pages. It allows the page to respond to user actions such as **clicks, form submissions, and mouse movements**. JavaScript can manipulate the content of a web page through **DOM (Document Object Model) manipulation**, handle events, and perform asynchronous operations using*

*technologies like **AJAX and the Fetch API**. This enables real-time updates without reloading the entire page, making applications more interactive and efficient.*

1.4 Browser Engine (Rendering Engine)

*The browser engine, also known as the **rendering engine**, is responsible for converting HTML, CSS, and JavaScript into a **visual display on the screen**. It processes HTML to build the **DOM (Document Object Model)** and CSS to create the **CSSOM (CSS Object Model)**. These structures are then combined to form the **render tree**, which is used to calculate the layout and paint the final output. The rendering engine ensures that web pages are displayed accurately and efficiently according to the defined structure and styles.*

2. Server-Side Technologies (Application Layer)

*Server-side technologies are executed on the **web server** and are responsible for **processing client requests, applying business logic, and generating dynamic responses**. They form the **application layer** of a web application, acting as a bridge between the client-side interface and the database. When a user sends a request, the server-side system processes it, interacts with the database if needed, and returns a dynamically generated web page. These technologies enable features such as **user authentication, data processing, session management, and real-time content generation**, making web applications interactive and data-driven.*

2.1 Server-Side Programming Languages

*Server-side programming languages are used to implement the **core logic of web applications**. They process user input, perform computations, and generate dynamic content. Common languages include PHP, Python, Java, and JavaScript (Node.js). These languages allow developers to build applications that can handle complex tasks such as form processing, user management, and communication with databases.*

2.1.1 PHP

*PHP (Hypertext Preprocessor) is a widely used **server-side scripting language** designed specifically for web development. It can be easily embedded within HTML and is commonly used to create dynamic web pages. PHP supports database connectivity, especially with MySQL, and is known for its simplicity and efficiency in building web applications.*

2.1.2 Python

*Python is a powerful and versatile programming language used in server-side development through frameworks such as **Django and Flask**. It is known for its simple syntax and readability,*

which makes development faster and more efficient. Python is widely used for building scalable and secure web applications.

2.1.3 Java (JSP & Servlets)

*Java is a robust and platform-independent language used for developing **enterprise-level web applications**. Technologies such as **JSP (JavaServer Pages)** and **Servlets** are used to create dynamic content on the server. Java provides high performance, security, and scalability, making it suitable for large-scale systems.*

2.1.4 Node.js

*Node.js is a server-side runtime environment that allows JavaScript to be executed on the server. It uses an **event-driven, non-blocking architecture**, which makes it highly efficient and suitable for building real-time applications. Node.js enables developers to use the same language (JavaScript) on both client and server sides.*

2.2 Application Frameworks

*Application frameworks provide a **structured environment** for developing web applications. They offer reusable components, libraries, and tools that simplify development and improve efficiency. Examples include **Django (Python), Spring (Java), and Express (Node.js)**. Frameworks help in managing code organization, security, and scalability.*

2.3 Template Engines

*Template engines are used to generate **dynamic HTML content** by combining static templates with dynamic data. They allow developers to separate the design (HTML) from the logic (server-side code). Examples include **EJS, Handlebars, and JSP**. Template engines improve code maintainability and readability.*

3. Data Layer (Database Technologies)

*The data layer is responsible for the **persistent storage, management, and retrieval of data** in a dynamic web application. It forms the foundation of data-driven systems by ensuring that information is stored efficiently and can be accessed whenever required. This layer interacts closely with the server-side technologies to provide accurate and real-time data for generating dynamic web pages.*

3.1 Relational Databases (RDBMS)

*Relational databases store data in the form of **tables consisting of rows and columns**, where relationships can be established between different tables. Examples include **MySQL and PostgreSQL**. These databases use **Structured Query Language (SQL)** to perform operations*

such as retrieving, inserting, updating, and deleting data. RDBMS ensures data integrity, consistency, and structured organization, making it suitable for applications requiring well-defined relationships.

3.2 NoSQL Databases

*NoSQL databases are designed to handle **unstructured or semi-structured data** and do not rely on a fixed table-based schema. An example is **MongoDB**, which stores data in flexible formats such as JSON-like documents. These databases provide **high scalability and flexibility**, making them suitable for modern applications that handle large volumes of diverse data.*

3.3 Data Access Mechanisms

Data access mechanisms define how applications interact with databases to perform operations and retrieve data efficiently.

3.3.1 ORM (Object Relational Mapping)

*ORM is a technique that **maps database tables to programming language objects**, allowing developers to work with data in an object-oriented manner instead of writing complex SQL queries. Examples include **Hibernate and Sequelize**. ORM improves productivity, reduces code complexity, and enhances maintainability.*

3.3.2 Query Processing

*Query processing involves executing database operations such as **SELECT, INSERT, UPDATE, and DELETE**. These operations are essential for retrieving existing data, adding new data, modifying records, and removing unwanted data. Efficient query processing ensures fast and reliable data access in dynamic web applications.*

4. Communication Technologies

*Communication technologies enable the **exchange of data between the client and server**, which is essential for dynamic web page functionality. They define how requests are sent and responses are received over the network.*

4.1 HTTP/HTTPS Protocol

*HTTP (HyperText Transfer Protocol) and HTTPS (secure version) define how **web clients and servers communicate**. They follow a **request-response model**, where the client sends a request and the server returns a response. These protocols are **stateless**, meaning each request is independent and does not retain previous information.*

4.2 AJAX (Asynchronous JavaScript and XML)

*AJAX allows web pages to **send and receive data asynchronously** without reloading the entire page. This improves performance and provides a smoother user experience by enabling partial updates of the web page, such as loading new data dynamically.*

4.3 APIs (Application Programming Interfaces)

*APIs enable **communication between different software systems**. They allow web applications to request and exchange data with external services. APIs commonly use **JSON format** for data transfer, making them efficient and easy to use in dynamic applications.*

5. Web Servers and Application Servers

Web servers and application servers are responsible for handling requests and executing application logic in dynamic web systems.

5.1 Web Servers

*Web servers handle **HTTP requests from clients** and deliver appropriate responses. They can serve both **static content (HTML, CSS)** and forward requests for dynamic processing. Examples include **Apache and Nginx**. Web servers ensure efficient request handling and resource management.*

5.2 Application Servers

*Application servers are responsible for executing **business logic and server-side processing**. They interact with databases, process user input, and generate dynamic content. Examples include **Tomcat and Node.js runtime**. These servers play a key role in building scalable and interactive applications.*

6. Supporting Technologies

*Supporting technologies assist in the **development, management, and optimization** of dynamic web applications.*

6.1 Version Control Systems

*Version control systems like **Git** are used to **track and manage changes in source code**. They allow multiple developers to collaborate efficiently and maintain different versions of the project.*

6.2 Package Managers

*Package managers such as **npm (Node Package Manager)** and **pip (Python Package Manager)** are used to **install, update, and manage external libraries and dependencies** required for development.*

6.3 Build Tools

*Build tools like **Webpack** are used to **optimize and bundle code** for production. They improve performance by reducing file sizes, managing assets, and preparing the application for deployment.*

2.2 HTML Programming

*HTML (HyperText Markup Language) is the **core structural language of the web**, used to define the **semantic organization and hierarchical structure** of web content. It is called a markup language because it uses **tags (markup)** to annotate content, describing its meaning and role rather than performing computation like a programming language. HTML forms the **foundation of all web pages**, upon which styling (CSS) and behavior (JavaScript) are layered.*

*At a deeper level, HTML is not just about displaying content—it is about creating a **structured document model** that browsers interpret and convert into a **visual and interactive interface**. Every HTML document is internally represented as a **tree structure (DOM)**, which allows dynamic manipulation and interaction.*

2.2.1 Basic Syntax and Structure

*HTML syntax defines how elements are written and organized to form a valid document. An HTML document follows a **well-defined hierarchical structure**, where elements are nested within one another, forming a **parent–child relationship** similar to a tree.*

Explanation of Structure

<!DOCTYPE html>
*The <!DOCTYPE html> declaration specifies that the document follows **HTML5 standards**. It helps the browser render the page in **standard mode**, avoiding compatibility issues. Without it, the browser may use outdated rendering rules. It ensures consistent display across different browsers.*

<html>
*The <html> element is the **root element** of the entire web page. It contains all other elements such as <head> and <body>. It forms the base of the **DOM structure** used by browsers. It defines the boundary of the complete HTML document.*

<head>
*The <head> section contains **metadata and configuration details** of the web page. It includes elements like <title>, <meta>, CSS links, and scripts. These elements control how the page behaves and appears. This content is not directly visible to users.*

<title>

*The <title> element defines the **name of the web page** shown on the browser tab. It helps users identify the page easily. It is also important for **search engine optimization (SEO)**. A clear title improves visibility and usability.*

<body>

*The <body> element contains all the **visible content** of the web page. It includes text, images, links, forms, and other interactive elements. This is the part users directly see and interact with. It represents the **display and interaction layer** of the page.*

2.3 Cascading Style Sheets (CSS)

*Cascading Style Sheets (CSS) is a **declarative style language** used to control the **visual presentation and layout** of web documents written in HTML. While HTML defines the **structure and meaning (semantics)** of content, CSS defines **how that content is rendered on the screen, printed, or displayed on different devices**.*

2.3.1 Introduction to CSS

*Cascading Style Sheets (CSS) is a **style sheet language** used to control the **presentation and layout** of web pages. While HTML is used to define the structure of a web page, CSS is responsible for determining how that content is **displayed visually** on the screen. It allows developers to apply styles such as **colors, fonts, spacing, alignment, and positioning** to HTML elements, thereby improving the overall appearance and user experience of a website.*

*CSS works on the principle of applying **style rules** to HTML elements. Each rule consists of a **selector**, which targets specific elements, and a **declaration block**, which defines the styling properties and their values. This approach enables precise control over the design of web pages. For example, developers can change the color of all headings, adjust spacing between elements, or control the layout of the entire page using CSS.*

*One of the most important features of CSS is the **separation of content and presentation**. By keeping HTML for structure and CSS for design, developers can maintain cleaner code and make changes easily without affecting the content. This also allows the same CSS file to be reused across multiple web pages, ensuring **consistency and efficiency** in web design.*

*The term **"cascading"** refers to the way CSS handles multiple style rules applied to the same element. When there are conflicting styles, CSS follows a priority system based on **specificity, inheritance, and source order** to determine which style should be applied. This makes CSS flexible and powerful in managing complex designs.*

*CSS also plays a key role in creating **responsive web designs**, allowing web pages to adapt to different screen sizes such as desktops, tablets, and mobile devices. With the help of techniques*

*like **media queries**, developers can ensure that web pages provide an optimal viewing experience across various devices.*

*In summary, CSS is an essential technology in web development that enhances the **visual presentation, usability, and responsiveness** of web pages. It transforms simple HTML content into **attractive and well-structured user interfaces**, making websites more engaging and user-friendly.*

2.3.2 Types of CSS

*CSS can be applied to HTML in three different ways: **Inline, Internal, and External CSS**. These types differ based on **where the styles are written, how they are applied, and their scope and priority**. Understanding these types is important for designing efficient and maintainable web pages.*

1. Inline CSS

Inline CSS is applied directly to an HTML element using the style attribute. It affects only that specific element and has the highest priority among all types of CSS. This method is useful for quick styling or testing small changes, but it mixes content with design, making the code difficult to maintain and reuse in larger projects.

Syntax & Example:

```
<p style="color: blue; font-size: 16px;">
   This is Inline CSS
</p>
```

---

2. Internal CSS

Internal CSS is defined within the <style> tag inside the <head> section of an HTML document. It is used to apply styles to all elements within a single web page. This approach provides better organization compared to inline CSS and allows consistent styling within the page, but it cannot be reused across multiple pages.

Syntax & Example:

```
<!DOCTYPE html>
<html>
<head>
   <style>
      p {
```

```
        color: green;
        font-size: 18px;
    }
  </style>
</head>
<body>
  <p>This is Internal CSS</p>
</body>
</html>
```

3. External CSS

External CSS is written in a separate .css file and linked to one or more HTML documents using the <link> tag. It allows styles to be applied across multiple pages, ensuring consistency and easy maintenance. This is the most efficient and widely used method in modern web development because it separates design from content and supports scalable applications.

Syntax & Example:

```
/* styles.css */
h1 {
  color: red;
}
```

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>This is External CSS</h1>
</body>
</html>
```

2.4 Dynamic Effects Using CSS

*CSS not only controls the appearance of web pages but also enables **dynamic visual effects** that enhance user interaction and experience. These effects are achieved using features like **hover effects, transitions, animations, positioning, and styling properties**. Dynamic effects make web pages more **interactive, responsive, and visually appealing** without requiring complex programming.*

2.4.1 Changing Text and Attributes

Changing text and attributes using CSS refers to modifying the ***appearance of text and visual properties of HTML elements dynamically*** *based on user interaction or predefined styles. CSS does not change the actual text content, but it can alter how the text looks, such as its* ***color, size, font, spacing, and decoration****. It can also modify element attributes like* ***background, border, and visibility****, making the page more interactive and visually appealing.*

This is commonly achieved using ***pseudo-classes*** *such as :hover, :focus, and :active, which apply styles when a user interacts with an element. For example, when a user moves the mouse over text, its color or size can change instantly, creating a dynamic effect without using JavaScript.*

Example:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p {
      color: black;
      background-color: lightgray;
      padding: 10px;
      width: 200px;
      text-align: center;
    }

    p:hover {
      color: white;
      background-color: blue;
      font-size: 20px;
      border: 2px solid black;
    }
  </style>
</head>
<body>

<p>Hover over me</p>

</body>
</html>
```

Explanation:

- *Initially, the paragraph text is **black with a light gray background**.*
- *When the user **moves the mouse over the text (hover)**:*
  - *Text color changes to **white***
  - *Background changes to **blue***
  - *Font size increases*
  - *A border appears around the element*

👉 *This demonstrates how CSS can dynamically change both **text properties** and **element attributes** using the :hover pseudo-class.*

2.4.2 Dynamic Style Changes

*Dynamic style changes in CSS refer to the ability to **modify the appearance of elements in response to user actions or events**. These changes are achieved using **pseudo-classes, transitions, and animations**, which allow elements to respond dynamically without the need for JavaScript. This makes web pages more **interactive, smooth, and visually engaging**.*

*CSS provides pseudo-classes such as :hover, :focus, and :active to detect user interactions. Additionally, properties like **transition** and **animation** help in creating smooth and gradual changes instead of instant effects. This enhances the overall **user experience** by making interactions feel natural and responsive.*

Example: Dynamic Style Change with Transition

```
<!DOCTYPE html>
<html>
<head>
  <style>
    button {
      background-color: blue;
      color: white;
      padding: 10px 20px;
      border: none;
      transition: 0.5s;
    }

    button:hover {
      background-color: green;
      transform: scale(1.1);
    }
```

```
  </style>
</head>
<body>

<button>Hover Me</button>

</body>
</html>
```

Explanation:

- *Initially, the button has a **blue background**.*
- *When the user hovers over the button:*
  - *Background changes to **green***
  - *Button slightly **increases in size***
- *The transition: 0.5s property ensures the change happens **smoothly** instead of instantly*

2.4.3 Text Graphics and Placement

*Text graphics and placement in CSS refer to the techniques used to **enhance the visual appearance of text** and to **control its position on a web page**. CSS provides various properties to style text, such as **color, font, size, alignment, spacing, and shadows**, making it more attractive and readable. At the same time, it allows precise control over where text appears using **layout and positioning properties**.*

*Text graphics involve improving the look of text using effects like **text-shadow, font styling, and decoration**, while placement involves arranging text using properties such as **text-align, margin, padding, and position**. These techniques help create well-structured and visually appealing web pages.*

Example: Text Styling and Placement

```
<!DOCTYPE html>
<html>
<head>
  <style>
    h1 {
      color: purple;
      text-align: center;
      text-shadow: 2px 2px 5px gray;
    }
```

```
    .box {
        position: absolute;
        top: 50px;
        left: 100px;
        font-size: 18px;
        color: blue;
    }
  </style>
</head>
<body>

<h1>Styled Heading</h1>
<div class="box">Placed Text</div>

</body>
</html>
```

Explanation:

- *The <h1> text is:*
  - *Center aligned*
  - *Colored purple*
  - *Given a shadow effect for better appearance*
- *The text:*
  - *Is positioned using position: absolute*
  - *Placed at a specific location (top and left)*

2.5 Multimedia Effects

*Multimedia effects in CSS represent the **visual transformation layer** of web design, where elements such as images, videos, and containers are enhanced using **rendering effects and time-based animations**. These effects operate at the **browser rendering level**, meaning they do not change the actual content but alter how it is **displayed to the user in real time**.*

From a conceptual perspective, multimedia effects combine two important ideas:

- ***Visual Transformation (Filters)** → Changes how an element looks*
- ***Temporal Transformation (Transitions)** → Changes how an element evolves over time*

*This allows developers to create **interactive, responsive, and visually rich interfaces** without relying on JavaScript.*

2.5.1 Filters

*CSS filters are part of the **post-processing stage of rendering**, where the browser applies graphical effects to an already rendered element. Instead of modifying the original image, filters act like a **layer of visual processing**, similar to effects in photo editing software.*

Types of Visual Transformations

- ***blur()*** *→ Reduces sharpness (simulates focus loss)*
- ***brightness()*** *→ Adjusts light intensity*
- ***contrast()*** *→ Enhances difference between light and dark*
- ***grayscale()*** *→ Removes color information*
- ***sepia()*** *→ Adds vintage brown tone*

Multiple filters can be combined:

```
filter: blur(2px) brightness(120%) contrast(150%);
```

Example with Concept

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
   width: 300px;
   filter: grayscale(100%) blur(2px);
   transition: 0.5s;
}

img:hover {
   filter: none;
}
</style>
</head>
<body>

<img src="image.jpg" alt="Sample">

</body>
</html>
```

2.5.2 Transitions

*Transitions represent **time-based interpolation of property values**. Instead of jumping from one state to another instantly, CSS calculates **intermediate values over time**, creating smooth animations.*

Transition Properties

- ***transition-property*** *→ What changes (e.g., width, color)*
- ***transition-duration*** *→ How long the change takes*
- ***transition-timing-function*** *→ Speed curve (ease, linear, ease-in-out)*
- ***transition-delay*** *→ When the change starts*

transition: property duration timing-function delay;

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 200px;
  height: 100px;
  background-color: blue;
  transition: all 0.5s ease;
}

div:hover {
  background-color: red;
  width: 300px;
}
</style>
</head>
<body>

<div></div>

</body>
</html>
```

3. JavaScript and Events

3.1 Introduction to JavaScript

*JavaScript is a **high-level, interpreted scripting language** used to add **interactivity and dynamic behavior** to web pages. While HTML provides structure and CSS handles presentation, JavaScript controls the **behavior of web applications**. It allows developers to create features such as **form validation, animations, event handling, and real-time updates**.*

*JavaScript is executed in the **web browser**, making it a core technology of web development. It can manipulate the **Document Object Model (DOM)**, enabling dynamic changes to content, structure, and styles after a page has loaded. JavaScript follows an **event-driven programming model**, where actions like clicks, key presses, and mouse movements trigger specific code execution.*

*In modern web development, JavaScript is also used beyond browsers (e.g., Node.js), but its primary role remains in building **interactive and responsive user interfaces**.*

3.1.1 Client-Side JavaScript

*Client-side JavaScript refers to JavaScript code that is executed **directly in the user's browser** rather than on the server. It is mainly used to enhance the **user interface and user experience** by enabling immediate responses to user actions without requiring communication with the server.*

Client-side JavaScript can perform tasks such as:

- ***Form validation** before submitting data*
- ***Dynamic content updates** without reloading the page*
- ***Handling user events** like clicks and keyboard input*
- ***Manipulating HTML and CSS using the DOM***
- ***Making asynchronous requests (AJAX)***

*Because it runs in the browser, client-side JavaScript provides **faster interaction** and reduces server load. However, it has limitations such as **security restrictions** and limited access to server resources.*

Example: Client-Side JavaScript

```
<!DOCTYPE html>
<html>
<head>
   <script>
      function showMessage() {
         alert("Hello, this is Client-Side JavaScript!");
      }
   </script>
</head>
```

```
<body>

<button onclick="showMessage()">Click Me</button>

</body>
</html>
```

Explanation:

- *When the user clicks the button, the function showMessage() is executed*
- *The browser displays an **alert message instantly***
- *No server communication is required*

3.1.2 Server-Side JavaScript

*Server-side JavaScript refers to JavaScript code that is executed on the **web server** instead of the user's browser. It is used to handle **request processing, business logic, and data management** in web applications. With the help of environments like **Node.js**, JavaScript can be used to build complete backend systems.*

*When a user sends a request (such as submitting a form or accessing a page), the server-side JavaScript processes the request, interacts with the **database if required**, and generates a **dynamic response**. This response is then sent back to the client's browser. Unlike client-side JavaScript, which focuses on user interaction, server-side JavaScript works behind the scenes to ensure that the application functions correctly.*

*Server-side JavaScript is commonly used for tasks such as **user authentication, data validation, database operations, file handling, and API development**. Since it runs on the server, it provides better **security and control**, especially for handling sensitive data.*

Example: Server-Side JavaScript (Node.js)

```
const http = require('http');

const server = http.createServer((req, res) => {
   res.write("Welcome to Server-Side JavaScript");
   res.end();
});

server.listen(3000);
```

Explanation

- *A server is created using JavaScript*

- *It listens for requests from the client (browser)*
- *When a request is received, it sends a response*
- *The processing happens on the **server side**, not in the browser*

3.1.3 Core Features of JavaScript

- *JavaScript is a **lightweight and interpreted language**, which makes it easy to use and execute directly in the browser.*
- *It is a **dynamically typed language**, so variables can hold different types of values without type declaration.*
- *JavaScript supports **event-driven programming**, where code runs based on user actions like clicks and key presses.*
- *It is an **object-based language**, allowing data and functions to be organized using objects.*
- *JavaScript is **platform independent**, meaning it works on any device with a browser or JavaScript engine.*
- *It can be **integrated with HTML and CSS** to dynamically modify web page content and design.*
- *JavaScript enables **client-side validation**, which checks user input before sending data to the server.*
- *It supports **asynchronous processing**, allowing multiple tasks to run without blocking execution.*

3.2 JavaScript Basics

*JavaScript basics include the fundamental concepts required to write and understand JavaScript programs. These concepts form the **foundation of programming in JavaScript**, such as variables, data types, operators, and control structures. Understanding these basics is essential for developing **interactive and dynamic web applications**. Among these, **data types and variables** are the most important, as they are used to store and manipulate data in a program.*

3.2.1 Data Types and Variables

*In JavaScript, **variables** are used to store data values, and **data types** define the type of data that a variable can hold. JavaScript is a **dynamically typed language**, which means you do not need*

to declare the data type of a variable explicitly; it is automatically determined based on the value assigned.

Variables in JavaScript

*Variables are declared using the keywords **var**, **let**, and **const**.*

- ***var** → Old method, function-scoped*
- ***let** → Block-scoped, commonly used*
- ***const** → Block-scoped, value cannot be changed*

Example:

```
let name = "John";
const age = 25;
var city = "Hyderabad";
```

Data Types in JavaScript

JavaScript supports different types of data, which are mainly classified into:

1. Primitive Data Types

*Primitive data types in JavaScript are the **basic, fundamental types of data** that store **single values directly in memory**. These data types are **immutable**, meaning their values cannot be changed once created, and they are stored by **value rather than reference**.*

- ***Number** → Represents numeric values*
- ***String** → Represents text*
- ***Boolean** → Represents true or false*
- ***Undefined** → Variable declared but not assigned*
- ***Null** → Represents empty value*

Example:

```
let x = 10;         // Number
let name = "Ram";   // String
let isValid = true; // Boolean
let y;              // Undefined
let z = null;       // Null
```

Non-primitive data types

*Non-primitive data types in JavaScript are **complex data types** that store **multiple values or collections of data**. Unlike primitive types, they are stored by **reference (memory address)***

rather than by value, and they are ***mutable****, meaning their contents can be changed after creation.*

Common non-primitive data types include ***Object, Array, and Function****, and they are used to represent structured and dynamic data in JavaScript programs.*

1. Object

An ***object*** *in JavaScript is a collection of* ***key-value pairs****, where each key (property) is associated with a value. Objects are used to represent real-world entities and group related data together.*

Example:

```
let person = {
   name: "Ram",
   age: 20,
   city: "Hyderabad"
};

console.log(person.name); // Ram
```

👉 *Here, person is an object with properties like name, age, and city.*

2. Array

An ***array*** *is a special type of object used to store* ***multiple values in a single variable****. The values are stored in an ordered list and accessed using an index (starting from 0).*

Example:

```
let numbers = [10, 20, 30, 40];

console.log(numbers[0]); // 10
```

👉 *Here, numbers is an array, and elements are accessed using index positions.*

3. Function

A ***function*** *is a block of code designed to perform a* ***specific task****. It can be reused and executed whenever needed.*

Example:

```
function greet() {
   return "Hello World";
}
console.log(greet()); // Hello World
```

👉 *Here, greet is a function that returns a message when called.*

3.2.2 Operators

*Operators in JavaScript are **special symbols or keywords** used to perform operations on **operands (values or variables)**. They form the core mechanism through which a program performs **computation, comparison, and decision-making**. Every operator takes one or more operands and produces a **result**, which can be a value or a Boolean outcome.*

*From a conceptual perspective, operators act as the **processing units of expressions**, enabling the transformation of input values into meaningful output. They are essential for building **logical conditions, mathematical calculations, and dynamic behaviors** in JavaScript programs.*

1. Arithmetic Operators (Mathematical Processing)

*Arithmetic operators are used to perform **basic mathematical operations** on numeric values. These operators are fundamental for calculations in programming.*

- *+ (Addition)*
- *- (Subtraction)*
- ** (Multiplication)*
- */ (Division)*
- *% (Modulus – remainder)*
- *** (Exponentiation)*

Example:

```
let a = 10, b = 3;
console.log(a + b);  // 13
console.log(a % b);  // 1
```

👉 *These operators transform numerical data through **mathematical computation**.*

2. Assignment Operators (Value Binding)

*Assignment operators are used to **assign values to variables**. They define how data is stored and updated during program execution.*

- *= ⟶ Assign value*
- *+= ⟶ Add and assign*
- *-= ⟶ Subtract and assign*
- **= /= %=*

Example:

```
let x = 5;
x += 3;  // x = x + 3 ⟶ 8
```

👉 *These operators combine **calculation + assignment**, making code more efficient.*

3. Comparison Operators (Relational Evaluation)

*Comparison operators are used to **compare two values** and return a Boolean result (true or false). They are mainly used in decision-making.*

- *== ⟶ Equal (value only)*
- *=== ⟶ Strict equal (value + type)*
- *!= ⟶ Not equal*
- *> < >= <=*

Example:

```
console.log(5 == "5");   // true
console.log(5 === "5");  // false
```

👉 *These operators perform **relational evaluation** between operands.*

4. Logical Operators (Decision Making)

*Logical operators are used to combine or modify **Boolean expressions**. They are essential for controlling program flow.*

- *&& (AND) ⟶ true if both conditions are true*
- *|| (OR) ⟶ true if at least one condition is true*
- *! (NOT) ⟶ reverses the condition*

Example:

```
let age = 20;
console.log(age > 18 && age < 30); // true
```

👉 *These operators enable **complex condition building**.*

5. Unary Operators (Single Operand Operations)

*Unary operators operate on **a single operand** and are used for simple modifications.*

- *++ ⟶ Increment*
- *-- ⟶ Decrement*
- *typeof ⟶ Returns data type*

Example:

```
let x = 10;
x++;
console.log(x); // 11
```

👉 *These operators perform **quick transformations on a single value**.*

6. Ternary Operator (Conditional Expression)

*The ternary operator is a **compact form of conditional statement (if-else)**. It evaluates a condition and returns one of two values.*

Syntax:

condition ? value1 : value2;

Example:

```
let marks = 40;
let result = (marks >= 35) ? "Pass" : "Fail";
```

👉 *It simplifies decision-making into **a single expression**.*

7. Type and Special Operators (Advanced Concepts)

JavaScript also includes special operators for advanced operations:

- *typeof → Returns data type*
- *instanceof → Checks object type*
- *delete → Removes object property*

Example:

```
let name = "Ram";
console.log(typeof name); // string
```

3.2.3 Expressions and Statements

*In JavaScript, **expressions and statements** are the basic building blocks of a program. They define how **operations are performed** and how **instructions are executed**. Understanding the difference between them is essential for writing correct and structured code.*

1. Expressions

*An **expression** is any valid combination of **values, variables, and operators** that produces a **result (value)**. Expressions are evaluated by the JavaScript engine to return a value.*

👉 *In simple terms:*

Expression → Evaluates → Produces a value

Examples:

```
5 + 3          // 8
x * 2          // depends on x
"Hello" + " JS" // "Hello JS"
```

👉 *All the above examples return a value, so they are expressions.*

Types of Expressions

- ***Arithmetic Expression*** → *10 + 5*
- ***String Expression*** → *"A" + "B"*
- ***Logical Expression*** → *x > 10 && y < 5*
- ***Assignment Expression*** → *x = 10*

2. Statements

*A **statement** is a complete instruction that performs an action. It does not necessarily return a value but tells the program **what to do**.*

👉 *In simple terms:*

Statement → Executes → Performs an action

Examples:

```
let x = 10;       // variable declaration
if (x > 5) {      // conditional statement
   console.log(x);
}
```

👉 *These are statements because they perform actions rather than just returning values.*

3.3 JavaScript Advanced Concepts

*JavaScript advanced concepts focus on powerful features that allow developers to build **modular, reusable, and efficient programs**. These include **functions, objects, arrays, and built-in objects like Date and Math**, which help in handling data, performing operations, and organizing code effectively. Understanding these concepts is essential for developing **real-world dynamic applications**.*

3.3.1 Functions

*A **function** in JavaScript is a reusable block of code designed to perform a **specific task**. It helps in organizing programs into smaller, manageable units, improving **readability, reusability, and modularity**. Functions are executed only when they are **called or invoked**, allowing the same code to be used multiple times without repetition.*

*From a conceptual perspective, a function acts like a **mapping mechanism** that takes **inputs (parameters)**, processes them, and produces an **output (return value)**.*

Basic Syntax of Function

```
function functionName(parameters) {
   // code to be executed
```

```
    return value;
}
```

Example:

```
function add(a, b) {
    return a + b;
}

console.log(add(4, 6)); // 10
```

👉 *Here, add is a function that takes two inputs and returns their sum.*

Advantages of Functions

- *Functions promote **code reusability** by allowing the same code to be used multiple times.*
- *They help in **modular programming** by dividing large programs into smaller parts.*
- *Functions improve **code readability and structure**.*
- *They make **maintenance easier**, as changes can be done in one place.*
- *Functions reduce **code duplication** and keep code clean.*
- *They simplify **debugging**, as errors can be found easily.*
- *Functions provide **abstraction** by hiding complex logic.*
- *They offer **flexibility** through parameters and return values.*
-

3.3.2 Objects

*An **object** in JavaScript is a **non-primitive data type** used to store data in the form of **key-value pairs**. It allows grouping of related data and functionality into a single unit, making programs more **organized and structured**. Objects are widely used to represent **real-world entities** such as a person, student, or product.*

Each object consists of:

- ***Properties** → Data (key-value pairs)*
- ***Methods** → Functions inside an object*

Basic Syntax

```
let objectName = {
    key1: value1,
    key2: value2
};
```

Example:

```
let student = {
   name: "Ram",
   age: 20,
   greet: function() {
      return "Hello " + this.name;
   }
};

console.log(student.name);      // Ram
console.log(student.greet());   // Hello Ram
```

👉 *Here, name and age are properties, and greet is a method.*

3.3.3 Arrays

*An **array** in JavaScript is a **non-primitive data type** used to store **multiple values in a single variable**. It allows data to be stored in an **ordered collection**, where each element is accessed using an **index number starting from 0**. Arrays are widely used when handling lists of data such as numbers, names, or objects.*

Basic Syntax

```
let arrayName = [value1, value2, value3];
```

Example:

```
let numbers = [10, 20, 30, 40];

console.log(numbers[0]); // 10
console.log(numbers[2]); // 30
```

👉 *Here, elements are accessed using their index positions.*

3.3.4 Date and Math Objects

*JavaScript provides built-in objects like **Date** and **Math** to perform **time-related operations and mathematical calculations** efficiently. These objects are part of the JavaScript standard library and help developers avoid writing complex logic for common tasks.*

1. Date Object

*The **Date object** is used to work with **dates and time** in JavaScript. It allows you to create, display, and manipulate date and time values.*

Creating a Date Object

let today = new Date();

👉 *This creates a Date object with the* ***current date and time****.*

Common Date Methods

- *getFullYear() → Returns the year*
- *getMonth() → Returns month (0–11)*
- *getDate() → Returns day of the month*
- *getDay() → Returns day of the week*
- *getHours() → Returns hours*
- *getMinutes() → Returns minutes*

Example:

```
let now = new Date();

console.log(now.getFullYear()); // e.g., 2026
console.log(now.getMonth());    // e.g., 2 (March)
console.log(now.getDate());     // e.g., 18
```

👉 *Used for applications like* ***clocks, calendars, and timestamps****.*

2. Math Object

The ***Math object*** *provides* ***mathematical functions and constants****. It does not require object creation and is used directly.*

Common Math Methods

- *Math.sqrt(x) → Square root*
- *Math.pow(x, y) → Power*
- *Math.random() → Random number (0–1)*
- *Math.floor(x) → Round down*
- *Math.ceil(x) → Round up*
- *Math.round(x) → Round to nearest integer*

Example:

```
console.log(Math.sqrt(16));   // 4
console.log(Math.random());   // random value
console.log(Math.floor(4.7)); // 4
```

Features

- *Date handles* ***time and calendar operations***
- *Math handles* ***numerical calculations***

- *Both are **built-in objects***
- *Improve efficiency and reduce manual coding*

3.4 Document Object Model (DOM)

*The **Document Object Model (DOM)** is a programming interface for web documents that represents an HTML or XML document as a **tree structure of objects**. Each element, attribute, and piece of text in the document is treated as a **node** in this tree. The DOM allows JavaScript to **access, modify, and manipulate** the structure, content, and style of a web page dynamically.*

*From a conceptual point of view, the DOM acts as a **bridge between HTML and JavaScript**, enabling developers to create **interactive and dynamic web pages**. When a web page is loaded, the browser converts the HTML into a **DOM tree**, which can then be manipulated using JavaScript.*

3.4.1 Introduction to DOM

*The DOM provides a structured representation of a document in the form of a **hierarchical tree**, where:*

- *The entire document is the **root node (document)***
- *HTML elements are **element nodes***
- *Text inside elements are **text nodes***
- *Attributes are also treated as nodes*

Example Structure:

```
<html>
  <body>
    <h1>Hello</h1>
    <p>Welcome</p>
  </body>
</html>
```

👉 *DOM Tree Representation:*

```
document
 └── html
     └── body
         ├── h1 → "Hello"
         └── p  → "Welcome"
```

Key Features of DOM

- *Represents HTML as a **tree structure***
- *Allows **dynamic modification** of content*
- *Supports **event handling***
- *Enables interaction between **JavaScript and HTML***

3.4.2 Manipulating Elements

*Manipulating elements in the DOM refers to the process of **accessing, modifying, creating, and removing HTML elements dynamically using JavaScript**. This is the core mechanism that enables web pages to become **interactive and responsive**.*

*At a deeper level, DOM manipulation means **changing the in-memory DOM tree**, which automatically reflects changes in the **rendered web page**.*

1. Selecting Elements (Access Phase)

Before manipulating an element, it must be accessed from the DOM.

Methods:

```
document.getElementById("id");
document.getElementsByClassName("class");
document.getElementsByTagName("p");
document.querySelector("selector");
```

👉 *These methods return a **reference to DOM nodes**.*

2. Changing Content (Data Manipulation)

Content of elements can be modified using:

```
element.innerHTML = "New Content";
element.textContent = "Text Only";
```

- *innerHTML → changes HTML content*
- *textContent → changes only text*

3. Changing Styles (Presentation Manipulation)

CSS properties can be modified dynamically:

```
element.style.color = "red";
element.style.backgroundColor = "yellow";
```

👉 *This updates the **CSSOM + Render Tree***

4. Changing Attributes (Attribute Layer)

Attributes define element behavior:

```
element.setAttribute("src", "image.jpg");
element.getAttribute("src");
```

👉 *Used for:*

- *Images (src)*
- *Links (href)*

5. Creating Elements (Expansion of DOM)

New elements can be created and inserted:

```
let newEl = document.createElement("p");
newEl.textContent = "New Paragraph";
document.body.appendChild(newEl);
```

👉 *This modifies the* ***DOM tree structure***

6. Removing Elements (Deletion)

```
element.remove();
```

👉 *Removes node from DOM → UI updates*

7. Traversing Elements (Navigation)

```
element.parentNode;
element.childNodes;
element.nextSibling;
```

👉 *Allows movement within the DOM tree*

8. Event-Based Manipulation (Dynamic Interaction)

```
button.onclick = function() {
   document.body.style.background = "blue";
};
```

3.5 Events and Event Handlers

Events and event handlers allow web pages to respond to ***user actions and system-generated activities****, making applications interactive. Each type of event serves a specific purpose in handling different kinds of user interactions.*

3.5.1 Introduction to Events

*An **event** is any action that occurs in the browser, such as a click, key press, or page load. JavaScript uses **event handlers** (functions) to respond to these actions. When an event is triggered, the corresponding handler executes code, enabling dynamic behavior on the web page.*

Example:

```
<button onclick="showMessage()">Click Me</button>
```

3.5.2 Mouse Events

*Mouse events occur when the user interacts with elements using a **mouse**, such as clicking, hovering, or moving the cursor. These events are commonly used to create interactive effects like highlighting, color changes, and tooltips.*

Example:

```
<p onmouseover="changeColor()" onmouseout="resetColor()">
   Hover over me
</p>

<script>
function changeColor() {
   document.body.style.background = "yellow";
}
function resetColor() {
   document.body.style.background = "white";
}
</script>
```

👉 *The background color changes when the mouse enters and leaves the text.*

3.5.3 Keyboard Events

Keyboard events are triggered when the user presses or releases keys on the keyboard. These events are useful for handling input fields, shortcuts, and real-time typing effects.

Example:

```
<input type="text" onkeyup="displayText(this.value)">

<script>
```

```
function displayText(value) {
   console.log(value);
}
</script>
```

👉 *The typed text is displayed in the console as the user types.*

3.5.4 Form Events

Form events occur when users interact with form elements such as input fields, buttons, and forms. These events are mainly used for ***validation and data handling*** *before submitting information to the server.*

Example:

```
<form onsubmit="return validate()">
   <input type="text" id="name">
   <input type="submit">
</form>

<script>
function validate() {
   let name = document.getElementById("name").value;
   if (name === "") {
      alert("Name required");
      return false;
   }
}
</script>
```

👉 *The form is not submitted if the input field is empty.*

4.PHP

4.1 Introduction to PHP

PHP (Hypertext Preprocessor) is a ***server-side scripting language*** *used for developing* ***dynamic and interactive web applications****. Unlike client-side languages such as JavaScript, PHP code is executed on the* ***web server****, and the output is sent to the user's browser in the form of HTML. This makes PHP suitable for handling tasks such as* ***form processing, database interaction, session management, and content generation****.*

*PHP is widely used because it is **open-source, platform-independent, and easy to integrate with HTML**. It can run on various operating systems like Windows, Linux, and macOS, and supports multiple web servers such as Apache and Nginx. One of the key advantages of PHP is its strong support for databases like **MySQL**, making it ideal for building **data-driven web applications**.*

*PHP scripts are embedded within HTML using special tags, allowing developers to mix **server-side logic with front-end structure**. When a user requests a PHP page, the server processes the PHP code, executes it, and sends the resulting output (usually HTML) back to the browser.*

Basic Syntax of PHP

```
<?php
   echo "Hello, World!";
?>
```

👉 *The <?php ?> tags indicate the start and end of PHP code.*

4.1 .1 Key Features of PHP

- *PHP is a **server-side scripting language**, which executes code on the server and sends output to the browser.*
- *It is **open-source and free**, making it widely accessible and cost-effective.*
- *PHP is **platform independent**, meaning it can run on Windows, Linux, and macOS.*
- *It can be easily **embedded within HTML**, allowing seamless integration of logic and content.*
- *PHP has strong **database support**, especially with MySQL, for building data-driven applications.*
- *It supports **session and cookie management**, enabling user tracking and authentication.*
- *PHP provides **fast performance** and efficient execution for web applications.*
- *It is **easy to learn and use**, making it suitable for beginners and professionals.*
- *PHP supports a wide range of **frameworks and libraries** for rapid development.*
- *It is highly **flexible and scalable**, suitable for both small and large web applications.*

4.1.2 PHP Syntax

*PHP syntax defines the **rules and structure** for writing PHP programs. A PHP script is written inside special tags and is executed on the **server**, with the output sent to the browser. PHP syntax is simple and similar to languages like C and Java, making it easy to learn.*

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>PHP Syntax Example</h2>

<?php
   $name = "Ram";         // variable declaration
   $age = 20;

   echo "Name: " . $name . "<br>";   // output
   echo "Age: " . $age;
?>

</body>
</html>
```

4.2 PHP Basics

*PHP basics include the fundamental concepts required to write and understand PHP programs. These concepts form the **foundation of server-side programming**, enabling developers to create **dynamic and interactive web applications**. PHP basics mainly cover **variables, data types, operators, control structures, and input/output handling**.*

*PHP is executed on the **server**, and its output is sent to the browser as HTML. Understanding these basic concepts is essential for working with **forms, databases, and dynamic content generation**.*

4.2.1 Variables and Data Types

*In PHP, **variables and data types** are fundamental concepts used to **store, represent, and manipulate data**. Variables act as containers for storing values, while data types define the **kind of data** a variable can hold. PHP is a **dynamically typed language**, which means there is no need to declare the data type explicitly; it is automatically determined based on the assigned value.*

1. Variables in PHP

*A **variable** is used to store data that can be used and modified throughout a program. In PHP, all variables begin with a **dollar sign ($)**, followed by the variable name.*

Rules for Variables:

- *Must start with $*
- *Must begin with a letter or underscore*
- *Cannot start with a number*
- *Case-sensitive*

Example:

```
<?php
   $name = "Ram";
   $age = 20;
   $price = 99.5;
?>
```

👉 *Here, $name, $age, and $price are variables storing different values.*

3. Data Types in PHP

*Data types in PHP define the **type of value** a variable can store and determine how that data is **handled, processed, and manipulated** during program execution. Since PHP is a **dynamically typed language**, the data type is automatically assigned based on the value given to the variable.*

Classification of Data Types

*PHP data types are broadly divided into **three categories**:*

- ***Scalar (Simple) Types***
- ***Compound Types***
- ***Special Types***

1. Scalar Data Types (Textbook Style – Deep & Clear)

*Scalar data types in PHP are the **basic data types** that store **a single value** at a time. They are the simplest form of data representation and are used for handling **individual pieces of information** such as numbers, text, and logical values. These data types are called "scalar" because they represent a **single unit of data**, not a collection.*

Data Type	***Definition***	***Example Syntax***	***Example Values***

Integer	*An integer is a data type used to store **whole numbers without decimal points**. It can represent positive, negative, or zero values.*	*$num = 100;*	*10, -5, 0*
Float (Double)	*A float is used to store **numbers with decimal points or fractional values**. It is suitable for precise calculations involving real numbers.*	*$price = 99.99;*	*3.14, -2.5*
String	*A string is a data type used to store a **sequence of characters or text**, enclosed within single or double quotes.*	*$name = "PHP";*	*"Hello", "World"*
Boolean	*A boolean represents a **logical value**, which can be either true or false. It is commonly used in conditional statements.*	*$isValid = true;*	*true, false*

4.2.2 Arrays and Strings

Compound Data Types

*Compound data types in PHP are used to store **multiple values in a single variable**. Unlike scalar data types, which hold only one value, compound types allow grouping of related data, making programs more **organized and efficient**. These data types are essential for handling **collections of data and complex structures**.*

Types of Compound Data Types

PHP mainly provides two compound data types:

1. Array

*An **array** is a data type used to store **multiple values in a single variable**. The values in an array are stored in an **ordered manner** and can be accessed using an **index or key**.*

```
<?php
   $colors = array("Red", "Green", "Blue");
?>
```

👉 *Each element in the array is accessed using its index, starting from 0.*

2. Object

*An **object** is a data type that stores data in the form of **properties and methods**. It is used to represent real-world entities and is created using **classes**.*

```
<?php
   class Student {
      public $name = "Ram";
   }
   $obj = new Student();
?>
```

👉 *Objects allow combining **data and functions** in a structured way.*

Special Data Types

*Special data types in PHP are used for **specific purposes** that go beyond storing simple or multiple values. These types are designed to handle **special situations**, such as representing the absence of a value or working with external resources like files and databases.*

Types of Special Data Types

PHP mainly provides two special data types:

1. NULL

*The **NULL** data type represents a variable that has **no value assigned**. It is used to indicate that a variable is empty or does not contain any data.*

```
<?php
   $x = null;
?>
```

👉 *A variable becomes NULL when:*

- *It is explicitly assigned null*
- *It is declared but not assigned a value*

👉 *Used for:*

- *Resetting variables*
- *Checking empty values*

2. Resource

*A **resource** is a special data type that holds a reference to an **external resource**, such as a file, database connection, or network stream. It is not a direct value but a **handle to access external data**.*

```
<?php
   $file = fopen("test.txt", "r");
?>
```

👉 *Here, $file is a resource representing an open file.*

👉 *Used for:*

- *File handling*
- *Database connections*
- *External data operations*

2. Strings in PHP

*A **string** in PHP is a data type used to store a **sequence of characters**, such as letters, numbers, and symbols. Strings are widely used for handling **textual data**, including names, messages, and content displayed on web pages. They are one of the most commonly used data types in PHP programming.*

*Strings can be defined using **single quotes (' ')** or **double quotes (" ")**, and PHP treats them slightly differently in terms of variable interpretation and escape sequences.*

Defining Strings

```
<?php
   $str1 = "Hello PHP";
   $str2 = 'Web Development';
?>
```

👉 *Both are valid strings.*

String Operations in PHP

Operation	***Description***	***Function / Operator***	***Example***	***Output***
Concatenation	*Combines two or more strings*	*.*	*$a." ".$b*	*Hello World*
String Length	*Returns number of characters in a string*	*strlen()*	*strlen("PHP")*	*3*

Word Count	Counts number of words in a string	str_word_count()	str_word_count("Hello PHP")	2
String Replace	Replaces text within a string	str_replace()	str_replace("PHP","JS","I love PHP")	I love JS
String Reverse	Reverses a string	strrev()	strrev("PHP")	PHP (reversed)
Uppercase	Converts string to uppercase	strtoupper()	strtoupper("php")	PHP
Lowercase	Converts string to lowercase	strtolower()	strtolower("PHP")	php

4.2.3 Operators and Expressions

*In PHP, **operators and expressions** are fundamental concepts used to perform **calculations, comparisons, and logical operations**. Operators are symbols that perform operations on variables and values, while expressions are combinations of **operands and operators** that produce a result.*

1. Operators in PHP

Operators are used to perform different types of operations such as arithmetic, assignment, comparison, and logical operations.

1. Operators in PHP

Operator Type	**Description**	**Operators**	**Example**	**Output**
Arithmetic	Performs mathematical calculations	+ - * / %	$a + $b	15
Assignment	Assigns values to variables	= += -= *= /=	$x += 5	15
Comparison	Compares two values and returns Boolean	== != > < >= <=	$a > $b	true
Logical	Combines multiple conditions	`&&		!`

2. Expressions in PHP

Expression Type	Description	Example	Result
Arithmetic Expression	Performs mathematical operation	5 + 3	8
Assignment Expression	Assigns value to variable	$x = 10	10
Comparison Expression	Compares values	$x > 5	true
Logical Expression	Combines conditions	$x > 5 && $x < 10	true

4.3 Control Structures

4.3.1 Conditional Statements

Conditional statements in PHP are used to **control the execution of a program based on specific conditions**. They enable decision-making by allowing different blocks of code to execute depending on whether a condition evaluates to **true or false**. These statements form the basis of **logical control** in programming and are widely used in real-world applications such as validation, authentication, and data processing.

Types of Conditional Statements in PHP

1. if Statement

The if statement is used to execute a block of code only when a specified condition is true. If the condition evaluates to false, the block is skipped. It is mainly used for simple decision-making.

```
<?php
   $age = 20;
   if ($age >= 18) {
      echo "Eligible to vote";
   }
?>
```

👉 If the condition is false, the block is skipped.

2. if-else Statement

The **if-else statement** provides two possible paths of execution based on a condition. If the condition is true, the if block executes; otherwise, the else block is executed. It is used when there are **two alternative outcomes**.

```
<?php
   $age = 16;
   if ($age >= 18) {
```

```
    echo "Adult";
  } else {
    echo "Minor";
  }
?>
```

👉 *One of the two blocks will always execute.*

3. if-elseif-else Statement

*The **if-elseif-else statement** is used to check multiple conditions sequentially. The first condition that evaluates to true will execute its block, and the rest are skipped. It is useful for handling **multiple decision scenarios**.*

```
<?php
  $marks = 75;

  if ($marks >= 90) {
    echo "Grade A";
  } elseif ($marks >= 60) {
    echo "Grade B";
  } else {
    echo "Grade C";
  }
?>
```

👉 *Conditions are checked one by one until one is true.*

4. switch Statement

*The **switch statement** is used to compare a variable against multiple values. It executes the block of code that matches the value of the variable. It is more efficient than multiple if-else statements when dealing with **fixed values**.*

```
<?php
  $day = "Monday";

  switch ($day) {
    case "Monday":
      echo "Start of week";
      break;
    case "Friday":
```

```
        echo "End of week";
        break;
    default:
        echo "Normal day";
  }
?>
```

4.3.2 Looping Statements

*Looping statements in PHP are control structures used to **execute a block of code repeatedly** based on a specified condition. They help in reducing code redundancy and improving efficiency by allowing the same set of instructions to run multiple times without rewriting the code.*

*Looping is essential in programming when working with **repetitive tasks**, such as processing arrays, generating sequences, or performing calculations multiple times.*

1. for Loop

*The **for loop** is a control structure used to execute a block of code repeatedly for a **specified number of iterations**. It is most suitable when the number of repetitions is **known before execution**. The loop consists of three parts: initialization, condition, and update.*

Syntax:

```
for(initialization; condition; update) {
   // code
}
```

Example:

```
<?php
for ($i = 1; $i <= 3; $i++) {
   echo $i;
}
?>
```

2. while Loop

*The **while loop** is used to repeatedly execute a block of code as long as a given condition remains **true**. The condition is evaluated **before each iteration**, so the loop may not execute at all if the condition is false initially.*

Syntax:

```
while(condition) {
   // code
}
```

Example:

```
<?php
$i = 1;
while ($i <= 3) {
   echo $i;
   $i++;
}
?>
```

3. do-while Loop

*The **do-while loop** is a variation of the while loop in which the code block is executed **at least once**, regardless of the condition. The condition is checked **after the execution** of the loop body.*

Syntax:

```
do {
   // code
} while(condition);
```

Example:

```
<?php
$i = 1;
do {
   echo $i;
   $i++;
} while ($i <= 3);
?>
```

4. foreach Loop

*The **foreach loop** is specifically designed to iterate over **arrays or collections**. It automatically retrieves each element one by one, making it easier to process data without using an index.*

Syntax:

```
foreach($array as $value) {
    // code
}
```

Example:

```
<?php
$colors = array("Red", "Green", "Blue");

foreach ($colors as $color) {
    echo $color;
}
?>
```

4.4 Functions and Forms

4.4.1 Functions in PHP

*A **function** in PHP is a block of code designed to perform a specific task and can be executed whenever it is called. Functions are used to reduce repetition in programs and to organize code into manageable sections. By using functions, large programs can be divided into smaller and simpler parts, making them easier to understand, test, and maintain.*

*In PHP, functions can be either **built-in functions** or **user-defined functions**. Built-in functions are already provided by PHP, such as strlen(), date(), and sqrt(). User-defined functions are created by the programmer according to the requirements of the application.*

A function is defined using the function keyword, followed by the function name, parentheses, and a block of statements enclosed in curly braces. The function name should be meaningful and follow naming conventions.

Syntax

```
function function_name($parameter1, $parameter2, ...) {
    // statements
    return value; // optional
}
```

*A function can accept input values called **parameters** or **arguments**. These parameters are specified within the parentheses during function definition and are passed when the function is called. Functions may also return a value using the return statement.*

Example

```
<?php
function add($a, $b) {
    $sum = $a + $b;
    return $sum;
}

$result = add(10, 20);
echo "Sum = " . $result;
?>
```

In this example, the function add() takes two parameters and returns their sum.

Types of Functions in PHP

1. ***Built-in Functions***
   These are predefined functions provided by PHP.
   Example: strlen(), count(), date()

2. ***User-defined Functions***
   These are functions created by the programmer.
   Example: add(), multiply()

Advantages of Functions

- ***Code Reusability:*** *A function can be used multiple times in a program.*
- ***Modularity:*** *Breaks a large program into smaller parts.*
- ***Easy Maintenance:*** *Errors can be easily detected and corrected.*
- ***Improved Readability:*** *Makes the program easier to understand.*

Types of Function Parameters in PHP

1. Formal and Actual Parameters

- ***Formal Parameters:*** *Variables declared in the function definition.*
- ***Actual Parameters (Arguments):*** *Values passed during the function call.*

```
function multiply($x, $y) { // formal parameters
    return $x * $y;
}

echo multiply(4, 5); // actual parameters
```

2. Default Parameters

Default parameters allow a function to assign a value automatically if no argument is provided during the function call. This makes the function more flexible.

```
<?php
function welcome($name = "User") {
    echo "Welcome " . $name;
}

welcome();
?>
```

If no argument is passed, "User" is used as the default value.

3. Passing Parameters by Value

*In PHP, parameters are passed **by value** by default. This means that a copy of the argument is passed to the function. Any changes made to the parameter inside the function do not affect the original variable.*

```
<?php
function changeValue($num) {
    $num = $num + 10;
}

$a = 5;
changeValue($a);
echo $a;
?>
```

Output: 5 (original value remains unchanged)

4. Passing Parameters by Reference

Parameters can also be passed by reference using the & symbol. In this case, the function works directly with the original variable, and any changes made inside the function will affect the original value.

```
<?php
function changeValue(&$num) {
    $num = $num + 10;
}

$a = 5;
changeValue($a);
```

```
echo $a;
?>
```

Output: 15

4.4.2 Form Handling

Form handling in PHP is the process of collecting, processing, and managing user input submitted through HTML forms. It is one of the most important features of dynamic web applications, as it allows interaction between the user and the server.

An HTML form is used to gather input such as text, passwords, selections, and files from the user. This data is then sent to the server, where PHP scripts process it and generate a response.

Form handling is the technique of capturing user input through HTML forms and processing that data on the server using PHP scripts.

Basic Structure of an HTML Form

An HTML form is created using the <form> tag. It contains input elements like text fields, radio buttons, checkboxes, and submit buttons.

```
<form action="process.php" method="post">
   Name: <input type="text" name="username"><br>
   <input type="submit" value="Submit">
</form>
```

Attributes of <form> Tag

- **action:** *Specifies the file (PHP script) where form data is sent*
- **method:** *Specifies how data is sent (GET or POST)*

Methods of Form Submission

1. GET Method

In the GET method, form data is appended to the URL as query parameters.

```
<form method="get" action="process.php">
   Name: <input type="text" name="name">
   <input type="submit">
</form>
```

Characteristics:

- *Data is visible in the URL*
- *Limited data size*
- *Less secure*
- *Can be bookmarked*

PHP Code to Handle GET

```
<?php
$name = $_GET['name'];
echo "Hello " . $name;
?>
```

2. POST Method

In the POST method, form data is sent in the HTTP request body.

```
<form method="post" action="process.php">
   Name: <input type="text" name="name">
   <input type="submit">
</form>
```

Characteristics:

- *Data is not visible in the URL*
- *Can send large amounts of data*
- *More secure than GET*
- *Cannot be bookmarked*

PHP Code to Handle POST

```
<?php
$name = $_POST['name'];
echo "Hello " . $name;
?>
```

Superglobal Variables in PHP

Superglobal variables in PHP are predefined associative arrays that are automatically created by the PHP engine and are accessible in every scope (global, local, functions, classes). They play a crucial role in data flow between client, server, and application logic, forming the backbone of web-based interaction.

At a deeper level, superglobals act as data carriers that hold information coming from:

- *User input (forms, URL)*
- *Server environment*
- *Session and cookies*

- File uploads

1. $_GET (URL Data Handling)

*The $_GET superglobal is used to collect data sent through the **URL query string**. It is visible in the browser and mainly used for **non-sensitive data transfer**.*

```
<?php
echo $_GET['name'];
?>
```

👉 *Example:*

page.php?name=Ram

2. $_POST (Form Data Handling)

*The $_POST superglobal collects data sent via **HTTP POST method**, usually through forms.*

```
<?php
echo $_POST['name'];
?>
```

3. $_REQUEST (Combined Input)

*$_REQUEST is a combination of **$_GET, $_POST, and $_COOKIE**.*

```
<?php
echo $_REQUEST['name'];
?>
```

4. $_SERVER (Server Environment Data)

*$_SERVER contains information about the **server, request, and execution environment**.*

```
<?php
echo $_SERVER['PHP_SELF'];
echo $_SERVER['SERVER_NAME'];
?>
```

5. $_SESSION (Session Management)

*$_SESSION is used to store data that persists across **multiple pages** during a user session.*

```
<?php
session_start();
```

```
$_SESSION['user'] = "Ram";
?>
```

6. $_COOKIE (Client Storage)

*$_COOKIE stores small data on the **user's browser**.*

```
<?php
setcookie("user", "Ram", time()+3600);
echo $_COOKIE['user'];
?>
```

Deep Insight:

- *Stored on client side*
- *Used for preferences, tracking*

7. $_FILES (File Upload Handling)

*$_FILES is used to handle **file uploads via forms**.*

```
<?php
print_r($_FILES);
?>
```

8. $_ENV (Environment Variables)

*$_ENV stores **environment-level variables**.*

```
<?php
print_r($_ENV);
?>
```

9. $GLOBALS (Global Scope Access)

*$GLOBALS is used to access **global variables inside functions**.*

```
<?php
$x = 100;

function test() {
   echo $GLOBALS['x'];
}
```

```
test();
?>
```

4.5 Database Connectivity

*Database connectivity in PHP is the process of **connecting a PHP application to a database** (such as MySQL) in order to **store, retrieve, update, and delete data**. It is a fundamental concept used in building **dynamic and data-driven web applications** like login systems, registration forms, and online shopping websites.*

*In simple terms, database connectivity allows a website to **communicate with a database**. PHP acts as an **intermediate layer** between the user and the database. When a user performs an action (like submitting a form), PHP processes the request, interacts with the database, and then sends the result back to the browser.*

Complete Step-by-Step Process

Step 1: Create Database and Table

Before connecting PHP, a database and table must be created using MySQL.

```
CREATE DATABASE mydb;

USE mydb;

CREATE TABLE users (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(50),
   email VARCHAR(50)
);
```

👉 *This step prepares the **data storage structure**.*

Step 2: Establish Connection (Connecting to MySQL)

*In this step, PHP connects to the MySQL server using **mysqli_connect()**.*

Syntax:

```
$conn = mysqli_connect(host, username, password, database);
```

Example:

```
<?php
$conn = mysqli_connect("localhost", "root", "", "mydb");

if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
} else {
   echo "Connection successful";
}
?>
```

👉 *This creates a **communication link** between PHP and database.*

Step 3: Write and Execute Query

*After connection, SQL queries are written and executed using **mysqli_query()**.*

Example (INSERT):

```
<?php
$sql = "INSERT INTO users (name, email) VALUES ('Ram', 'ram@gmail.com')";
mysqli_query($conn, $sql);
?>
```

Example (SELECT):

```
<?php
$sql = "SELECT * FROM users";
$result = mysqli_query($conn, $sql);
?>
```

👉 *Queries are used to **manipulate and retrieve data**.*

Step 4: Handle and Fetch Results

*The result returned from the query is processed using functions like **mysqli_fetch_assoc()**.*

Example:

```
<?php
while ($row = mysqli_fetch_assoc($result)) {
   echo "Name: " . $row['name'] . "<br>";
}
?>
```

👉 *Each row is fetched as an **associative array**.*

Step 5: Close Connection

After completing operations, the database connection should be closed.

```
<?php
mysqli_close($conn);
?>
```

👉 *This frees resources and improves performance.*

4.6 Sessions and Cookies

*In web applications, communication between the browser and server happens through HTTP, which is a **stateless protocol**. This means the server does not remember previous requests. To overcome this limitation, PHP provides **sessions and cookies**, which help in **storing and maintaining user information across multiple web pages**.*

*Sessions and cookies are widely used in applications such as **login systems, shopping carts, and user preferences**, where it is necessary to remember user data during interaction.*

4.6.1 Session Handling

*A **session** is a method used to store user data on the **server side**. When a user visits a website, a unique **session ID** is created by the server and assigned to that user. This ID is used to identify and retrieve the user's stored data during their interaction with the website.*

*Session handling includes starting a session, storing data, accessing data, and destroying the session when it is no longer needed. Since the data is stored on the server, sessions are considered **more secure** and are mainly used for sensitive information such as login details.*

Simple Example:

```
<?php
session_start();
$_SESSION['user'] = "Ram";
echo $_SESSION['user'];
?>
```

👉 *This stores and displays user data using a session.*

4.6.2 Cookies

*A **cookie** is a small piece of data stored on the **user's browser**. It is used to remember information about the user, such as preferences or previous activity, even after the user leaves the website.*

Cookies are created by the server and sent to the browser, where they are stored for a specific period. When the user revisits the website, the browser sends the cookie back to the server, allowing the website to recognize the user.

*Cookies are less secure than sessions because they are stored on the client side, but they are useful for storing **non-sensitive data**.*

Simple Example:

```
<?php
setcookie("user", "Ram", time() + 3600);
echo $_COOKIE['user'];
?>
```

👉 *This stores and retrieves data using a cookie.*

Difference Between Sessions and Cookies

- *Sessions store data on the **server**, while cookies store data on the **browser***
- *Sessions are **more secure**, cookies are **less secure***
- *Sessions are usually **temporary**, cookies can be stored for a longer time*
- *Sessions are used for **sensitive data**, cookies are used for **user preferences***

4.7 File Handling

*File handling in PHP refers to the process of **creating, reading, writing, updating, and deleting files** stored on the server. It is an important feature used to **store data permanently**, especially when a database is not required.*

*In simple terms, file handling allows a PHP program to **interact with files on the server**, just like working with documents on a computer. It is commonly used for **logging data, saving user input, and managing text files**.*

4.7.1 File Operations

*File operations in PHP refer to the various actions performed on files, such as **opening, reading, writing, appending, and closing files**. These operations allow a program to **store and manage data permanently** on the server. File handling is especially useful when working with **text files, logs, and simple data storage systems**.*

In PHP, file operations are performed using a set of built-in functions that provide ***efficient and controlled access to files****.*

1. Opening a File

Opening a file is the first step in file handling, which allows the program to access the file. In PHP, the fopen() function is used to open a file in a specific mode such as read or write. It prepares the file for further operations like reading or writing. Without opening, no file operation can be performed.

```
<?php
$file = fopen("data.txt", "r");
?>
```

2. Reading a File

Reading a file means retrieving data stored inside the file. PHP provides functions like fread() to read the content of a file and display it. This operation is useful when we want to access stored information. It allows programs to process and use file data.

```
<?php
$file = fopen("data.txt", "r");
echo fread($file, filesize("data.txt"));
fclose($file);
?>
```

3. Writing to a File

Writing to a file is the process of storing data into a file. The fwrite() function is used to write new content into a file. If the file already contains data, it may be overwritten depending on the mode. This operation is used to save new information permanently.

```
<?php
$file = fopen("data.txt", "w");
fwrite($file, "Hello PHP");
fclose($file);
?>
```

4. Appending to a File

Appending is used to add new data to an existing file without removing the previous content. It uses append mode (a) to ensure existing data is preserved. This operation is useful for updating logs or adding records. It helps in maintaining continuous data storage.

```
<?php
$file = fopen("data.txt", "a");
fwrite($file, "New Data");
fclose($file);
?>
```

5. Closing a File

Closing a file is the process of releasing the file after completing operations. The fclose() function is used to close an open file. It helps in freeing system resources and preventing errors. Proper closing ensures efficient file handling.

```
<?php
fclose($file);
?>
```

6. Deleting a File

Deleting a file means removing it permanently from the server. In PHP, the unlink() function is used for this purpose. This operation is useful when files are no longer needed. It helps in managing storage and avoiding unnecessary files.

```
<?php
unlink("data.txt");
?>
```

4.7.2 Reading and Writing Files

*Reading and writing files in PHP are important operations used to **retrieve data from files and store data into files**. These operations allow programs to handle **permanent data storage** without using a database. They are commonly used for **logs, text processing, and saving user input**.*

*In simple terms, **reading** means getting data from a file, and **writing** means saving data into a file.*

🔷 Reading Files

*Reading a file means accessing and displaying the content stored inside it. PHP provides functions like fread() and fgets() to read data from a file. This operation is useful when we want to **view or process existing data**.*

Example:

```
<?php
$file = fopen("data.txt", "r");
echo fread($file, filesize("data.txt"));
fclose($file);
?>
```

👉 *This reads and displays the entire file content.*

🔷 ***Writing Files***

Writing a file means storing new data into a file. PHP uses the fwrite() function to write content. If the file does not exist, it will be created automatically. Writing can also overwrite existing data depending on the mode used.

Example:

```
<?php
$file = fopen("data.txt", "w");
fwrite($file, "Hello PHP");
fclose($file);
?>
```

👉 *This writes data into the file.*

🔷 ***Appending Data***

Appending is a type of writing where new data is added to the end of the file without removing existing content.

Example:

```
<?php
$file = fopen("data.txt", "a");
fwrite($file, "New Line");
fclose($file);
?>
```

👉 *This adds data to the file.*

4.7.3 Directory Handling

Directory handling in PHP refers to the process of ***creating, opening, reading, and deleting directories (folders)*** *on the server. A directory is a container used to* ***organize files in a structured manner****, just like folders on a computer.*

In simple terms, directory handling allows PHP programs to ***manage folders and the files inside them****, which is useful for applications like file uploads, storage systems, and content management.*

Types of Directory Operations

1. Creating a Directory

Creating a directory means making a new folder on the server. PHP uses the mkdir() function for this purpose.

```
<?php
mkdir("myfolder");
?>
```

👉 *This creates a new directory named* ***myfolder****.*

2. Opening a Directory

Opening a directory allows PHP to access and read its contents. The opendir() function is used.

```
<?php
$dir = opendir("myfolder");
?>
```

👉 *This prepares the directory for reading files.*

3. Reading a Directory

Reading a directory means listing the files inside it. The readdir() function is used to fetch each file.

```
<?php
while (($file = readdir($dir)) !== false) {
   echo $file . "<br>";
}
closedir($dir);
?>
```

👉 *This displays all files in the directory.*

4. Closing a Directory

After operations, the directory should be closed using closedir() to free resources.

```
<?php
closedir($dir);
?>
```

5. Deleting a Directory

A directory can be removed using the rmdir() function. The directory must be empty before deletion.

```
<?php
rmdir("myfolder");
?>
```

👉 *This deletes the directory.*

5.1 XML (Extensible Markup Language)

5.1.1 Introduction to XML

*XML (Extensible Markup Language) is a **standard markup language used for storing, organizing, and transporting data** in a structured format. It is designed to be **self-descriptive**, meaning the data is clearly defined by user-created tags. XML is widely used in web technologies for **data exchange between different systems**, especially when applications are built using different platforms.*

*XML does not focus on how data is displayed; instead, it focuses on **how data is structured and described**. This makes XML different from HTML, which is mainly used for presentation.*

*XML is a **text-based, extensible markup language** that allows users to define their own tags to represent and store data in a structured and platform-independent manner.*

🔶 ***Basic Structure of XML***

*An XML document follows a **tree-like hierarchical structure**, consisting of elements and tags.*

```
<?xml version="1.0"?>
<student>
   <name>Ram</name>
   <age>20</age>
</student>
```

👉 *<student> → Root element*

👉 *<name>, <age> → Child elements*

Characteristics of XML

- *XML is **extensible**, allowing users to create their own custom tags.*
- *It is **self-descriptive**, as tags clearly define the data they contain.*
- *XML is **platform independent**, meaning it can be used across different systems.*
- *It follows a **structured (hierarchical) format**, organizing data in a tree-like structure.*
- *XML is both **human-readable and machine-readable**.*
- *It is widely used for **data exchange between different applications**.*
- *XML follows **strict syntax rules**, ensuring well-formed documents.*
- *It supports **separation of data and presentation**.*
- *XML is **flexible and scalable**, suitable for small and large data.*

Rules of XML (Well-Formed XML)

- *An XML document must contain **only one root element**, which acts as the top-level container for all other elements.*
- *All XML tags must be **properly closed**, either with a closing tag or as a self-closing tag.*
- *XML is **case-sensitive**, so opening and closing tags must match exactly in case.*
- *Elements must be **properly nested**, meaning tags should be closed in the correct order without overlapping.*
- *All attribute values in XML must be **enclosed within quotation marks** (either single or double quotes).*

Advantages of XML

- *XML is extensible because it allows users to **create their own custom tags** according to their requirements.*
- *XML is self-descriptive, as the tags clearly **define the meaning of the data** they contain.*
- *XML is platform independent, which means it can be **used across different systems without compatibility issues**.*
- *XML supports data exchange by enabling **communication between different applications and technologies**.*
- *XML provides a structured format that helps in **organizing and managing data efficiently**.*
- *XML is both human-readable and machine-readable, making it **easy to understand and process**.*
- *XML allows separation of data from presentation, which improves **flexibility in application design**.*
- *XML is flexible and scalable, so it can **handle both small and large amounts of data effectively**.*

5.1.2 Creating XML Documents

*Creating an XML document means **representing data in a structured and organized way using custom-defined tags**. XML does not have fixed tags like HTML; instead, it allows users to define their own tags based on the type of data they want to store. This makes XML highly flexible and suitable for **data storage and data exchange**.*

*In simple terms, an XML document is created by **writing data inside tags and arranging it in a hierarchical (tree-like) structure**.*

🔷 Basic Structure of XML Document

*An XML document usually begins with a declaration and must contain a **single root element** that holds all other elements.*

```
<?xml version="1.0"?>
<root>
   <data>Example</data>
</root>
```

🔶 Steps to Create an XML Document

1. Write XML Declaration

The XML declaration is placed at the top of the document and specifies the version and encoding used. It helps the system understand how to process the XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
```

2. Define Root Element

Every XML document must have one root element, which acts as the main container for all other elements. All data must be placed inside this root element.

```
<students>
</students>
```

3. Add Child Elements

Child elements are placed inside the root element to represent actual data. These elements can also contain sub-elements, forming a nested structure.

```
<students>
   <student>
      <name>Ram</name>
```

```
    <age>20</age>
  </student>
</students>
```

4. Maintain Proper Structure

While creating an XML document, it is important to follow correct rules such as closing tags properly, using correct nesting, and maintaining case sensitivity. This ensures the document is ***well-formed****.*

🔷 ***Example of Complete XML Document***

```
<?xml version="1.0"?>
<library>
  <book>
    <title>Web Technology</title>
    <author>Ram</author>
  </book>
</library>
```

5.1.3 XML Style Sheets

XML Style Sheets are a mechanism used to ***define how XML data is presented, processed, and transformed****. While XML is designed to* ***store and transport data****, it does not specify how that data should appear. XML Style Sheets solve this limitation by providing a way to* ***separate data (content) from its presentation (format and structure)****.*

At an advanced level, XML Style Sheets are not only used for styling but also for ***data transformation, restructuring, filtering, and logical processing****, making them a powerful tool in data-driven systems.*

Types of XML Style Sheets

1. CSS (Cascading Style Sheets)

CSS is used for ***basic visual formatting*** *of XML elements. It controls appearance such as colors, fonts, and layout but does not modify the structure of XML.*

Working Principle:

👉 *CSS maps XML elements → visual styles*

Example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="style.css"?>

<student>
   <name>Ram</name>
   <age>20</age>
</student>

student {
   display: block;
   color: blue;
}
```

Types of CSS

CSS can be applied in different ways based on ***scope, reusability, and level of control****. The three main types are Inline CSS, Internal CSS, and External CSS.*

🔷 *1. Inline CSS*

Inline CSS is a method of applying styles directly within an HTML element using the style attribute. It is used to control the appearance of a ***single element only****. This type of CSS has the* ***highest priority*** *in the cascade. However, it is not suitable for large projects because it mixes content with design and reduces maintainability.*

Syntax:

```
<tag style="property: value;">
```

Example:

```
<p style="color: red; font-size: 18px;">Hello World</p>
```

🔷 *2. Internal CSS*

Internal CSS is defined within the <style> tag inside the <head> section of an HTML document. It is used to apply styles to ***all elements within a single web page****. This method improves organization compared to inline CSS but still limits reusability. It is suitable for* ***small or single-page applications****.*

Syntax:

```
<style>
selector {
```

```
    property: value;
}
</style>
```

Example:

```
<html>
<head>
<style>
p {
    color: blue;
    font-size: 16px;
}
</style>
</head>
<body>
<p>Hello World</p>
</body>
</html>
```

3. External CSS

External CSS is written in a separate .css file and linked to HTML documents using the <link> tag. It allows styles to be applied across ***multiple web pages****, ensuring consistency and maintainability. This method follows the principle of* ***separating content and presentation****, making it ideal for large and scalable web applications.*

Syntax:

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

Example:

HTML File:

```
<link rel="stylesheet" href="style.css">
<p>Hello World</p>
```

CSS File (style.css):

```
p {
    color: green;
```

```
  font-size: 20px;
}
```

2. XSL (Extensible Stylesheet Language)

*XSL (Extensible Stylesheet Language) is a language used to **define how XML data should be displayed, formatted, and transformed**. While XML is used to store data, XSL is used to **present that data in a structured and user-friendly format**, such as HTML.*

*In simple terms, XSL acts as a **bridge between XML data and its final output**, helping convert raw XML into readable and meaningful content.*

XML File:

```
<students>
  <student>
    <name>Ram</name>
  </student>
</students>
```

XSL File:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>Student Name</h2>
<xsl:value-of select="students/student/name"/>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

👉 *This converts XML data into an HTML page.*

5.2 XML Processing

*XML Processing refers to the techniques used to **read, access, modify, and extract data from XML documents**. Since XML stores data in a structured format, special methods are required to **navigate and work with that data efficiently**.*

*In simple terms, XML processing helps programs to **understand and manipulate XML data** for different purposes such as display, storage, and data exchange.*

5.2.1 XML DOM (Document Object Model)

*XML DOM (Document Object Model) is a programming interface that represents an XML document as a **tree-like structure of nodes**. In this model, every part of the XML document—such as elements, attributes, and text—is treated as an individual **node**.*

*This allows programs to **access, navigate, modify, add, or delete data** in the XML document easily. XML DOM is widely used in languages like **JavaScript and PHP** for dynamic data handling.*

Basic Concept

👉 ***XML Document → Converted into DOM Tree (Nodes)***

```
<student>
  <name>Ram</name>
</student>
```

*XML DOM is a model that represents an XML document as a **hierarchical tree of nodes**, where each element is treated as an object. It allows developers to **access and manipulate XML data programmatically**. Through DOM, elements can be **read, updated, or deleted dynamically**. It provides a structured way to work with XML documents.*

5.2.2 XML Query Language

*XML Query Language is used to **search, locate, and extract specific data from an XML document**. Since XML files may contain large amounts of structured data, it is not efficient to read the entire document every time. Query languages help in selecting **only the required elements or information**.*

*In simple terms, XML Query Language allows us to **ask questions to an XML document and get the needed data**.*

*XML Query Language is a method used to **retrieve and filter data from XML documents**. It helps in selecting specific elements, attributes, or values based on conditions. The most commonly used XML query language is **XPath**, which navigates through the XML structure. It improves efficiency by extracting only relevant data.*

Example XML:

```
<students>
  <student>
    <name>Ram</name>
    <age>20</age>
  </student>
</students>
```

Example Query (XPath):

/students/student/name

👉 *This selects the <name> element from the XML document.*

Key Features of XML Query Language

- *Used to **locate specific data in XML***
- *Supports **filtering and conditions***
- *Works with **XML DOM and XSLT***
- *Improves efficiency by **retrieving only required data***
- *Can access both **elements and attributes***

5.3 JSP (Java Server Pages)

*JSP (Java Server Pages) is a **server-side technology used to create dynamic web pages using Java**. It allows embedding Java code into HTML, enabling the generation of **interactive and data-driven content**. JSP works within the **client-server architecture**, where requests are processed on the server and results are sent to the browser.*

◇ *5.3.1 Introduction to JSP*

*JSP is a server-side scripting technology that enables developers to build **dynamic web applications using Java**. It allows the integration of Java code within HTML pages using special JSP tags. JSP pages are processed on the server and converted into servlets before execution. It simplifies web development by separating **presentation and business logic**.*

5.3.2 JSP Page Structure

*A JSP page consists of different components that define how the page is processed and displayed. It combines **static content (HTML)** with **dynamic elements (Java code)**.*

Main Elements

- ***Directives** → Instructions to JSP engine*
- ***Scriptlets** → Java code blocks*
- ***Expressions** → Output values*
- ***Declarations** → Variables and methods*

- ***HTML*** → *Page structure*

Example:

```
<%@ page language="java" %>
<html>
<body>

<%! int x = 10; %>

<%
int y = 20;
%>

Result: <%= x + y %>

</body>
</html>
```

5.3.3 JSP Processing

JSP Processing refers to the sequence of steps through which a JSP (Java Server Pages) file is executed on the server to generate a dynamic response. Instead of sending the JSP file directly to the browser, the server processes it internally and produces an HTML response.

1. Client Request

The process begins when a user sends a request for a JSP page through a web browser. This request is received by the web server, which identifies that the requested resource is a JSP file.

2. JSP Translation

In this stage, the server checks whether the JSP page has already been converted into a servlet. If not, the JSP file is translated into a ***Java servlet source code****. This conversion is necessary because JSP cannot be executed directly.*

3. Compilation Phase

The generated servlet source code is then compiled using a Java compiler. This results in a ***bytecode file (.class file)****, which can be executed by the server.*

4. Class Loading

After compilation, the servlet class is loaded into the server's memory. This prepares the servlet for execution and handling client requests.

5. Execution Phase

The servlet is executed to process the client's request. During this phase, all the embedded Java code within the JSP is executed, and the required dynamic content is generated.

6. Response Generation

Finally, the servlet generates output in the form of HTML. This output is sent back to the client's browser, where it is displayed as a web page.

Important Concept

The translation and compilation steps occur only once for a JSP file unless it is modified. For subsequent requests, the already compiled servlet is reused, which improves performance.

5.4 JSP Components

*JSP components are the **building blocks of JSP pages**. They are used to embed Java code, display output, and control page behavior.*

🔷 *5.4.1 Directives*

Directives are instructions given to the JSP engine about how to process the page. They control overall page behavior, such as importing classes or setting configurations. Directives do not produce output but affect the structure of the JSP page. They are written using <%@ %> syntax.

Example:

```
<%@ page language="java" %>
```

🔷 *5.4.2 Expressions*

*Expressions are used to **display output directly on the web page**. They evaluate Java expressions and convert the result into a string, which is then sent to the browser. Expressions simplify output generation without writing full Java code blocks.*

Example:

```
<%= 10 + 20 %>
```

👉 *Output: 30*

5.4.3 Code Snippets (Scriptlets)

Code snippets, also called scriptlets, are blocks of Java code written inside JSP pages. They are used to perform logic such as calculations, loops, and conditions. Scriptlets are executed on the server before the response is sent. They are written using <% %> tags.

Example:

```
<%
int x = 5;
int y = 10;
out.println(x + y);
%>
```

5.5 Advanced JSP Concepts

*Advanced JSP concepts provide the essential mechanisms required to build **dynamic, interactive, and data-driven web applications**. These concepts focus on how JSP manages **data, user interaction, application logic, and database communication**. They help in organizing code, improving reusability, and maintaining user state across multiple requests.*

5.5.1 Implicit Objects

*Implicit objects are predefined objects that are **automatically created by the JSP container** for every request. They allow direct access to important components such as request data, response handling, session information, and application context. These objects do not need to be explicitly declared in the JSP page. They simplify development by providing ready-to-use features.*

*Implicit objects act as a **link between the JSP page and the server environment**. They help in handling input from users, sending output to the browser, and managing application data. For example, the request object stores user input, while the session object keeps track of user information across pages.*

Common Implicit Objects

- *request → Handles client request data*
- *response → Sends response to browser*
- *session → Stores user session data*
- *application → Shares data across application*
- *out → Outputs content to browser*

Example (JSP Code)

```
<%
String name = request.getParameter("username");
session.setAttribute("user", name);
%>

Welcome <%= session.getAttribute("user") %>
```

5.5.2 Java Beans

*Java Beans are reusable Java classes used in JSP to **store data and perform business logic**. They follow a standard structure with private variables and public getter and setter methods. Java Beans help separate the logic from the presentation layer. This makes applications more organized and easier to maintain.*

*Java Beans represent the **data-handling component** of a web application. Instead of writing complex logic inside JSP pages, the logic is placed inside Beans and reused whenever needed. This approach improves clarity and supports modular programming.*

Example (Java Bean Class)

```
public class Student {
   private String name;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

Example (JSP Usage)

```
<jsp:useBean id="obj" class="Student" />

<jsp:setProperty name="obj" property="name" value="Ram" />

Name: <jsp:getProperty name="obj" property="name" />
```

Explanation of Example

- *<jsp:useBean> creates an object of the Java Bean*
- *<jsp:setProperty> assigns value to the Bean*

- *<jsp:getProperty> retrieves and displays the value*

5.5.3 Session Tracking

*Session tracking is a technique used to **maintain user-specific data across multiple requests** in a web application. Since HTTP does not remember previous interactions, session tracking is required to preserve user information. It allows continuous interaction between the user and the application. It is commonly used in login systems and user sessions.*

When a user interacts with a website, a unique session is created for that user. The server assigns a session ID and uses it to track the user's activities. This ensures that the application remembers user actions such as login status or selected items.

5.5.4 Database Connectivity

*Database connectivity in JSP refers to the process of **connecting a JSP application with a database** to store and retrieve data. It is commonly implemented using JDBC (Java Database Connectivity). This enables the application to perform operations like inserting, updating, and retrieving data. It is essential for dynamic applications.*

JSP interacts with the database through JDBC to process user requests. When a user submits data, it is stored in the database, and when required, it is retrieved and displayed. This allows applications to work with real-time data and provide dynamic content.