CMR INSTITUTE OF TECHNOLOGY	USN										

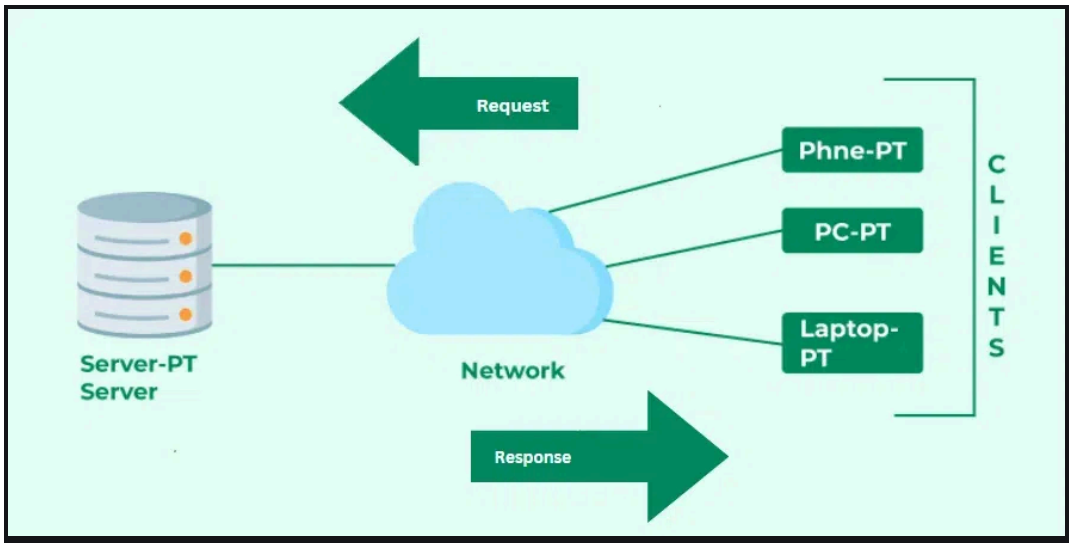
VTU Examination August - 2025				
Sub:	**Web Application Development**	**Code:**	**MMC205**	
	Answer Key	**Marks**	**OBE**	
			CO	RBT
	Module-1			
Q1.a	Explain the concept of client-server architecture with an example. **Client-Server Model/Architecture:-** The **Client-Server Model** is a distributed application architecture that divides tasks or workloads between **servers** (providers of resources or services) and **clients** (requesters of those services). In this model, a **client** sends a request to a **server** for data, which is typically processed on the server side. The server then returns the requested data to the client. **Clients** generally do not share resources with each other, but instead rely on the server to **provide the resources** or **services requested**. Common examples of the client-server model include email systems and the **World Wide Web** (WWW), where email clients interact with mail servers, and web browsers request resources from web servers. **How Does the Client-Server Model Work?** In this article, we are going to take a dive into the **Client-Server** model and have a look at how the **Internet** works via web browsers. This article will help us have a solid WEB foundation and help us easily work with WEB technologies. **Client** When we talk about a "Client," it refers to a **device** (usually a computer, smartphone, or application) that requests and receives services from a server. The client is the entity that initiates communication, asking for data or resources from the server. For instance, **web browsers** like **Google Chrome**, **Mozilla Firefox**, or **Safari** are common client applications that request data from a server to render web pages.	6	L2	CO1

Server

A **Server**, on the other hand, is a **remote computer or system** that provides data, resources, or services to clients. It listens to incoming client requests, processes them, and sends the required information back. A server can handle multiple client requests simultaneously.

For example, **Web servers** host websites, and **database servers** store and serve databases for applications. **In simple terms**, the client sends a request to the server, and the server serves the request as long as the data or service is available in its system.

Client Server Model



How the Browser Interacts With the Servers?

The process of interacting with servers through a browser involves several steps. Here's a breakdown of the steps taken when you enter a URL in a browser and receive the website data:

1. User Enters the URL (Uniform Resource Locator): The user types a website address (e.g., www.example.com) into the browser's address bar.

2. DNS (Domain Name System) Lookup: The browser sends a request to the **DNS server** to resolve the human-readable URL into an IP address (since computers use IP addresses to identify and connect to each other).

3. DNS Server Resolves the Address: The DNS server looks up the domain name and returns the **IP address** of the web server hosting the requested website.

4. Browser Sends HTTP/HTTPS Request: The browser sends an **HTTP/HTTPS request** to the IP address of the web server to fetch the website's data. HTTP

	(HyperText Transfer Protocol) or HTTPS (the secure version) is the protocol used for communication between the browser (client) and the web server (server). **5. Server Sends Website Files**: The server processes the request and sends the necessary website files (HTML, CSS, JavaScript, images, etc.) back to the browser. **6. Rendering the Website**: The browser **renders the files** and displays the website to the user. This rendering process involves several components:Together, these components, known as **Just-In-Time (JIT) Compilers**, allow the browser to convert raw data into a visual webpage. ● **DOM (Document Object Model) Interpreter**: Processes the HTML structure. ● **CSS Interpreter**: Applies styles to the HTML elements. ● **JS Engine**: Executes JavaScript code for interactivity. **Advantages of the Client-Server Model** **The Client-Server model offers several advantages that make it popular in networked and distributed systems:** ● **Centralized Data Management**: All data is stored in a centralized server, which makes it easier to manage, update, and back up. ● **Cost Efficiency**: Since the server handles most of the processing, clients require fewer resources and can be simpler devices, reducing costs. ● **Scalability**: Both **clients** and **servers** can be scaled separately. Servers can be upgraded to handle more clients, and new clients can be added without significant changes to the server infrastructure. ● **Data Recovery**: Centralized data storage on the server allows for better **data recovery** and easier **backup** strategies. ● **Security**: **Security measures** such as firewalls, encryption, and authentication can be centralized on the server, ensuring that sensitive data is protected. **Disadvantages of Client-Server Model** ● **Clients Are Vulnerable**: Clients are prone to viruses, Trojans, and worms if present in the Server or uploaded into the Server. ● **Servers Are Targets**: Servers are prone to Denial of Service (DOS) attacks, where the server is overwhelmed with traffic and made unavailable to legitimate clients.			
Q1.b	Differentiate between front-end and back-end development. **1. Definition**	4	L2	CO1

- **Front-End Development**:
  Refers to the part of a website or application that users interact with directly. It deals with the **user interface (UI)** and **user experience (UX)**.

- **Back-End Development**:
  Refers to the part that works **behind the scenes**. It handles **server-side operations**, databases, and business logic to ensure the front-end works smoothly.

2. Focus Area

- **Front-End**: Looks, design, and behavior of the application (what users see).

- **Back-End**: Functionality, data management, and logic (how the system works).

3. Technologies Used

- **Front-End**:
  - Languages: **HTML, CSS, JavaScript**
  - Frameworks/Libraries: **React, Angular, Vue.js, Bootstrap**
- **Back-End**:
  - Languages: **Java, Python, PHP, Node.js, Ruby, C#**
  - Frameworks: **Django, Flask, Spring, Express.js, Laravel**
  - Databases: **MySQL, MongoDB, PostgreSQL, Oracle**

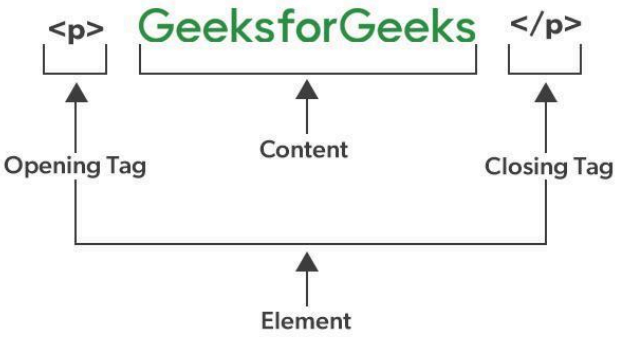
4. Role in Client-Server Model

- **Front-End**: Acts as the **client side**, sending requests and displaying responses.

- **Back-End**: Acts as the **server side**, processing requests, fetching/storing data, and sending responses.

---

5. Example

	Imagine an **online shopping site**: • **Front-End**: The product listings, search bar, shopping cart button, and checkout page that users see and interact with. • **Back-End**: The system that stores product details, manages inventory, processes payments, and updates order status in the database.			
Q1.c	List any five HTML5 elements and explain their purpose. **HTML Elements** An HTML Element consists of a start tag, content, and an end tag, which together define the element's structure and functionality. Elements are the basic building blocks of a webpage and can represent different types of content, such as text, links, images, or headings. For example, the element for paragraphs includes opening and closing tags with text content in between.  Syntax: <tagname >Your Contents... </tagname> HTML Element Code Example: In this example is a starting tag, is an ending tag and it contains some content between the tags, which form an element *<!-- HTML code to illustrate HTML elements -->* <!DOCTYPE html> <html>	5	L2	CO1

```
<head>
   <title>HTML Elements</title>
</head>
<body>
   <p>Welcome to GeeksforGeeks!</p>
</body>
</html>
```

Some Key Points About HTML Elements

1. Syntax:

- An opening tag indicates where the content begins: <tagname>.
- A closing tag indicates where the content ends: </tagname>.
- The actual content resides between the opening and closing tags.

2. Case Sensitivity:

- HTML tags are not case-sensitive. For example, <B> and <b> both represent bold text.
- However, it's a best practice to use lowercase tags for consistency and readability.

Block-Level Elements and Inline Elements

In HTML, elements are broadly categorized into two main types based on how they display in the document layout: block-level elements and inline elements.

1. Block-Level Elements – Block-level elements typically start on a new line and take up the full width available to them, regardless of their actual content width. This means they stack vertically and can contain other block-level elements as well as inline elements. Here are some examples of block-level elements:

Examples:

- : A general-purpose container for other elements.
- <p>: Defines a paragraph.
- <h1>, <h2>, …, <h6>: Heading elements of different levels.
- , <ul>: Ordered and unordered lists.
- <table>: Defines a table.
- <form>: Used for HTML forms to collect user inputs.
- <section>, <article>, <nav>, <aside>, <header>, <footer>: Semantic elements that define areas of a webpage.

2. **Inline Elements** – Inline elements do not start on a new line; they appear on the same line as adjacent content, as long as there is space. They only take up as much

width as their content requires. Inline elements are typically used within block-level elements to add content or style. Here are some examples of inline elements:

Examples:

- : A general-purpose inline container for phrasing content.
- <a>: Creates hyperlinks.
- <img>: Embeds an image.
  - , <b>: Used for strong emphasis and bold text, respectively.
- , <i>: Used for emphasis and italic text, respectively.
-
: Inserts a line break within text.

Components of Attribute
An HTML attribute consists of two primary components:

1. **attribute_name**: This is the name of the attribute, which specifies what kind of additional information or property you are defining for the element. Common attribute names include href, src, class, id, etc.

2. **attribute_value:** The value is assigned to the attribute to define the specific setting or behavior. It is always placed in quotes.

Types of HTML Attributes
HTML attributes can be broadly categorized based on their function and the type of elements they modify.

1.href

Definition:
Specifies the URL of the page the hyperlink points to. Used with the <a> tag.

Syntax:
Link Text
Example:
Visit Google

2. src

- Definition:
  Specifies the file path or URL of the image or media file to be displayed. Used with <img>, <video>, and <audio>.

  Syntax:
<img src="image.jpg">

- Example:
  <img src="logo.png" alt="Site Logo">

3. width and height

Definition:
Define the width and height of an image or media element in pixels.

Syntax:
<img src="image.jpg" width="200" height="150">
Example:
<img src="flower.jpg" width="300" height="200">

4. alt

Definition:
Provides alternative text for an image if it cannot be displayed. Also helps with accessibility.

Syntax:
<img src="image.jpg" alt="Description">
Example:
<img src="tree.png" alt="A green tree">

5. style

- Definition:
  Applies inline CSS styling to an HTML element.

  Syntax:
  <element style="property:value;">
  Example:
  Styled text

6. lang Attribute

Definition:
Declares the language of the web page's content. Used with the <html> tag.

Syntax:
<html lang="language_code">
Example:
<html lang="en">

7. title Attribute

- Definition:
  Provides additional information about an element. The info appears

as a tooltip when the mouse hovers over it.
Syntax:
<element title="Tooltip text">

- Example:
Hover over this paragraph

HTML5 Semantics

HTML5 introduced a range of semantic elements that clearly describe their purpose in human and machine-readable language. Unlike non-semantic elements, which provide no information about their content, semantic elements clearly define their content.

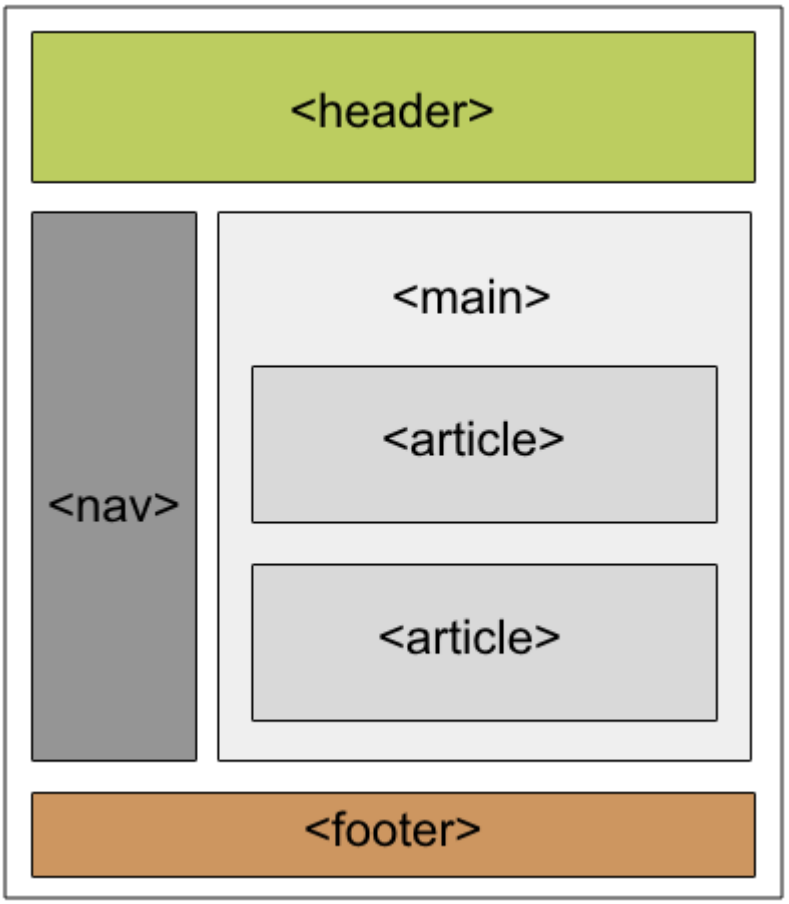
For instance, <form>, <table>, and <article> tags clearly define the content and purpose, to the browser.

Why Use Semantic HTML Tags?

- Accessibility: Semantic elements make web pages more accessible. Screen readers and other assistive technologies can interpret the structure and navigate the content more efficiently.
- Maintainability: Semantic HTML helps create a logically structured document, which is easier to read and maintain.

Semantic Elements

Here are some of the fundamental HTML5 semantic elements that you should use to structure your web content:



The following tags have been introduced for better structure −

- section − This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.
- article − This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.
- aside − This tag represents a piece of content that is only slightly related to the rest of the page.
- header − This tag represents the header of a section.
- footer − This tag represents a footer for a section and can contain information about the author, copyright information, et cetera.
- nav − This tag represents a section of the document intended for navigation.
- Mark − This tag can be used to mark up a conversation.
- figure − This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

Q1.d	Explain how to embed an audio file in a web page using HTML5. **HTML5 <audio> Tag** HTML5 introduced the <audio> element, which allows us to play audio directly in the browser **without plugins** (like Flash).	5	L2	CO1

---

2. Basic Syntax

```
<audio controls>
  <source src="audiofile.mp3" type="audio/mpeg">
  <source src="audiofile.ogg" type="audio/ogg">
  Your browser does not support the audio element.
</audio>
```

3. Explanation

- **<audio>**: The main tag for embedding audio.
- **controls**: Adds play, pause, volume, and seek bar controls.
- **<source>**: Specifies audio files. Multiple formats ensure cross-browser compatibility.
    - src → file path or URL of the audio.
    - type → audio format (e.g., audio/mpeg, audio/ogg).
- **Fallback Text**: Shown if the browser doesn't support <audio>.

4. Attributes of <audio>

- **controls** → Shows audio player controls.
- **autoplay** → Starts playing automatically when the page loads.
- **loop** → Plays the audio repeatedly.
- **muted** → Starts the audio in muted mode.
- **preload** → Hints how the browser should load audio:
    - auto → Load entire audio (default).
    - metadata → Load only audio details (duration, etc.).
    - none → Don't load audio until the user plays it.

5. Example with Attributes

	<audio controls autoplay loop muted> <source src="song.mp3" type="audio/mpeg"> <source src="song.ogg" type="audio/ogg"> Your browser does not support the audio element. </audio>			
	OR			
Q2.a	Explain the use of the <canvas> element in HTML5. How is it different from SVG? Illustrate with examples. **1. What is <canvas>?** ● The <canvas> element in HTML5 provides an **area (a drawing surface)** where you can draw graphics using **JavaScript**. ● It is mainly used for: ○ Drawing shapes (rectangles, circles, lines). ○ Creating charts and graphs. ○ Rendering images. ○ Developing games and animations. **2. Basic Syntax** <canvas id="myCanvas" width="400" height="200" style="border:1px solid black;"></canvas> ● id → Unique identifier (to access it via JavaScript). ● width & height → Size of the drawing area. ● Inside <canvas> → fallback text (shown if browser doesn't support canvas). **3. Drawing Example (Rectangle in Canvas)** <!DOCTYPE html> <html> <head> <title>Canvas Example</title> </head> <body> Canvas Rectangle	10	L2	CO1

```
  <canvas id="myCanvas" width="300" height="150" style="border:1px solid black;"></canvas>

  <script>
    // Access the canvas
    var c = document.getElementById("myCanvas");
    var ctx = c.getContext("2d");  // 2D drawing context

    // Draw a blue rectangle
    ctx.fillStyle = "blue";
    ctx.fillRect(50, 40, 200, 80);
  </script>
</body>
</html>
```

Difference Between <canvas> and SVG

Feature	<canvas> (HTML5 Canvas)	SVG (Scalable Vector Graphics)
Definition	Bitmap-based drawing surface.	XML-based vector graphics format.
Drawing Method	Drawn using JavaScript (imperative).	Defined using tags (declarative).
Scalability	Loses quality when scaled (pixel-based).	Infinite scalability (vector-based).
Interactivity	Requires manual coding (hit detection).	Built-in DOM elements, easy to style & interact.
Use Cases	Games, animations, charts with frequent redraws.	Static graphics, logos, maps, icons.
Performance	Faster for complex, dynamic graphics.	Better for simpler, scalable images.

Q2.b	List any three HTML5 APIs and explain their functionality briefly. **What is Web API?** API stands for Application Programming Interface. An API is some kind of interface that includes a set of functions and subroutines that allow	10	L2	CO1

programmers to access specific features or data of an application, operating system or other services.

A Web API is an application programming interface for the Web.

HTML APIs

All browsers have a set of built-in Web APIs to support complex operations, and to help accessing data.

Here are some of the main HTML5 APIs:

1. Geolocation API - This API is used to access the current location of a user (with latitude and longitude).

2. Drag and Drop API - This API enables you to use drag-and-drop features in browsers.

3. Web Storage API - This API has mechanisms to let browsers store key/value pairs (in a more intuitive way than cookies).

4. Web Workers API - This API allows a JavaScript to run in the background, without affecting the performance of the page. Users can continue to do whatever they want: clicking, selecting things, etc., while the web worker runs in the background.

5. Server-Sent Events API - This API allows a web page to automatically get updates from a server.

6. Canvas API - This API lets you draw graphics, on the fly, via JavaScript.

HTML Geolocation API

The Geolocation API is used to get the user's current location.

Locate the User's Position

The Geolocation API is used to access the user's current location.

Since this can compromise privacy, the location is not available unless the user approves it.

Using HTML Geolocation API

The Geolocation API is accessed via a call to **navigator.geolocation.** This will cause the browser to ask the user for permission to access their location data. If

the user accepts, the browser will search for the best available functionality on the device to access this information (for example GPS).

The getCurrentPosition() method is used to return the user's current location.

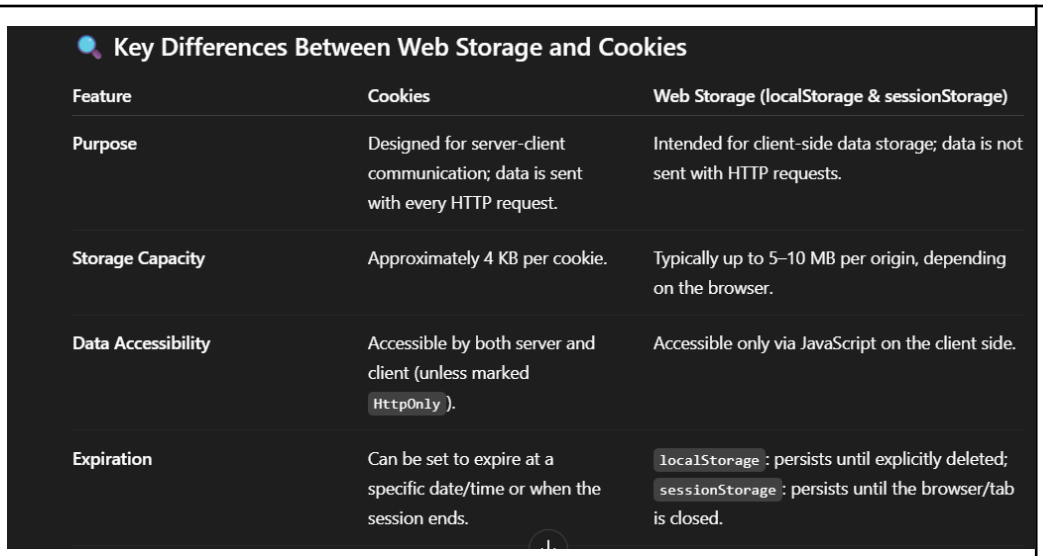
The example below returns the latitude and longitude of the user's current location:

Example

```
<script>
const x = document.getElementById("demo");
function getLocation() {
  if (navigator.geolocation) {
     navigator.geolocation.getCurrentPosition(success, error);
  } else {
     x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
function success(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
function error() {
  alert("Sorry, no position available.");
}
</script>
```

Example explained:

- Check if Geolocation is supported
- If Geolocation is supported, run the getCurrentPosition() method. If not, display a message to the user
- The success() function outputs the user's location in latitude and longitude
- The error() function alerts a text if the browser retrieves an error in getCurrentPosition()



🔍 Key Differences Between Web Storage and Cookies

Feature	Cookies	Web Storage (localStorage & sessionStorage)
Purpose	Designed for server-client communication; data is sent with every HTTP request.	Intended for client-side data storage; data is not sent with HTTP requests.
Storage Capacity	Approximately 4 KB per cookie.	Typically up to 5–10 MB per origin, depending on the browser.
Data Accessibility	Accessible by both server and client (unless marked `HttpOnly`).	Accessible only via JavaScript on the client side.
Expiration	Can be set to expire at a specific date/time or when the session ends.	`localStorage`: persists until explicitly deleted; `sessionStorage`: persists until the browser/tab is closed.

What is HTML Web Storage?

With web storage, applications can store data locally within the user's browser.

Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.

Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

Web Storage API Objects

Web storage provides two objects for storing data in the browser:

- **window.localStorage** - stores data with no expiration date (data is not lost when the browser tab is closed)
- **window.sessionStorage** - stores data for one session (data is lost when the browser tab is closed)

Test Web Storage API Support

Before using web storage, we can quickly check browser support for localStorage and sessionStorage:

Example

```
<script>
```

```
const x = document.getElementById("result");
if (typeof(Storage) !== "undefined") {
  x.innerHTML = "Your browser supports Web storage!";
} else {
  x.innerHTML = "Sorry, no Web storage support!";
}
</script>
```

The localStorage Object

The localStorage object stores the data with no expiration date. The data will not be lost when the browser is closed, and will be available the next day, week, or year.

Example

Use localStorage to set and retrieve name and value pairs:

```
<script>
const x = document.getElementById("result");

if (typeof(Storage) !== "undefined") {
  // Store
  localStorage.setItem("lastname", "Smith");
  localStorage.setItem("bgcolor", "yellow");
  // Retrieve
  x.innerHTML = localStorage.getItem("lastname");
  x.style.backgroundColor = localStorage.getItem("bgcolor");
} else {
  x.innerHTML = "Sorry, no Web storage support!";
}
</script>
```

Example explained:

- Use the localStorage.setItem() method to create name/value pairs
- Use the localStorage.getItem() method to retrieve the values set
- Retrieve the value of "lastname" and insert it into an element with id="result"
- Retrieve the value of "bgcolor" and insert it into the style backgroundColor of the element with id="result”

The sessionStorage Object

The sessionStorage object is equal to the localStorage object, except that it stores the data for only one session! The data is deleted when the user closes the specific browser tab.

Counting Clicks with sessionStorage

The following example counts the number of times a user has clicked a button, in the current session:

Example

```
<script>
function clickCounter() {
  const x = document.getElementById("result");
  if (typeof(Storage) !== "undefined") {
    if (sessionStorage.clickcount) {
      sessionStorage.clickcount = Number(sessionStorage.clickcount)+1;
    } else {
      sessionStorage.clickcount = 1;
    }
      x.innerHTML = "You have clicked the button " + sessionStorage.clickcount +
time(s) in this session!";
  } else {
    x.innerHTML = "Sorry, no Web storage support!";
  }
}
</script>
```

What is a Web Worker?

When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is an external JavaScript file that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

Web workers are useful for heavy code that can't be run on the main thread, without causing long tasks that make the page unresponsive.

- **Create a Web Worker Object**

- Once we have created the .js web worker file, we can call it from an HTML page.

	● The following lines checks if a worker (w) already exists, if not - it creates a new web worker object and points to the .js file: "demo_workers.js":
	
	`if (typeof(w) == "undefined") {`
	`w = new Worker("demo_workers.js");`
	`}`
	
	● Then we can SEND and RETRIEVE messages from the web worker.
	
	● Data is sent between web workers and the main thread via a system of messages - both sides send their messages using the postMessage() method, and respond to messages via the onmessage event handler.
	
	● Add an onmessage event listener to the web worker object.
	
	● onmessage
	
	● When the web worker in the .js posts a message, the code within the event listener is executed. The data from the web worker is stored in event.data.			

	Module-2			

Q3.a	What are CSS3 properties? List any five properties with their use.
	
	❖ **CSS Properties:-**
	● **<u>Font</u>**
	CSS font properties allow you to style text on web pages by controlling aspects like font family, size, weight, style, and spacing. Here's a detailed overview of the font-related properties in CSS:
	
	1. Font-family
	
	Specifies the typeface of the text. It can include multiple fonts as a fallback mechanism. If the browser doesn't support the first font, it will try the next.
	
	`p {`
	`font-family: "Arial", "Helvetica", sans-serif;`
	`}`
	
	● **Generic Families**:
	○ serif (e.g., Times New Roman)
	○ sans-serif (e.g., Arial)
	○ monospace (e.g., Courier)
		7	L2	CO2

- cursive (e.g., Comic Sans)

2. font-size

Defines the size of the text. It can be specified in:

- Absolute units: px, pt, cm, etc.
- Keywords: small, medium, large, etc.

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

3. font-weight

Controls the boldness of text. Acceptable values:

- Keywords: normal, bold, lighter, bolder
- Numerical values: 100 (thin) to 900 (extra-bold)

```
p {
        font-weight: bold;
}
h2 {
        font-weight: 300; /* Light */
    }
```

4. font-style

Defines the style of the font. Options:

- normal: Default text style
- italic: Italicized text
- oblique: Slanted text

```
h1 {
  font-style: italic;
}
```

5. font-variant

Used to enable or disable small-caps text (small uppercase letters).

```
p {
  font-variant: small-caps;
}
```

6. font-shorthand

The font property allows you to define multiple font properties in one line. The order is important:
font: font-style font-variant font-weight font-size/line-height font-family;

```
p {
   font: italic small-caps bold 16px/1.5 "Georgia", serif;
}
```

color:-

In CSS, colors are used to define the visual appearance of elements, such as text, backgrounds, borders, and more. CSS provides several ways to specify colors, offering flexibility for design and accessibility.

1. CSS Color Properties

These properties are commonly used to apply colors:

- **color**: Sets the color of text

```
p {
   color: red;
}
```

background-color: Sets the background color of an element.

```
div {
   background-color: lightblue;
}
```

border-color: Sets the color of an element's border.

```
div {
   border: 2px solid black;
   border-color: green;
}
```

2. Color Value Types

a. Named Colors

CSS has 140+ predefined color names.

```
h1 {

   color: blue;

}
```

Examples: red, green, blue, yellow, orange, purple, black, white, etc.

b. Hexadecimal Colors

Hexadecimal values represent colors using the format #RRGGBB.

```
p {
   color: #ff5733; /* Bright orange */
}
```

- **Short form**: #RGB (e.g., #f53 is equivalent to #ff5533).

c. RGB Colors

Specifies colors using red, green, and blue values (0–255).

```
body {
   background-color: rgb(255, 87, 51); /* Bright orange */
}
```

RGB with Alpha Transparency (rgba): Adds an alpha channel for transparency (0.0 to 1.0).

```
div {
   background-color: rgba(255, 87, 51, 0.5); /* 50% transparent */
}
```

Text Properties

CSS text properties are used to style and control the appearance and alignment of text on a webpage. Here's a comprehensive guide to the most commonly used CSS text properties:

1. color

Sets the color of the text.

```
p {
   color: blue;
}
```

2. text-align

Specifies the horizontal alignment of text within an element.
Values:

- left (default)
- right
- center
- justify

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

3. text-decoration

Adds or removes text decorations.
Values:

- none: Removes all decorations.
- underline: Adds an underline.
- overline: Adds a line above the text.
- line-through: Adds a strike-through line.
- blink (deprecated, not supported in most browsers).

```
a {
   text-decoration: overline;
}
```

4. text-transform

Controls the capitalization of text.
Values:

- none: Default (no transformation).
- capitalize: Capitalizes the first letter of each word.
- uppercase: Converts text to uppercase.
- lowercase: Converts text to lowercase.

```
h2 {
   text-transform: uppercase;
}
```

	5. letter-spacing
	
	Adjusts the spacing between characters (kerning).
	Values:
	
	● Use length units (px, em, %, etc.).
	
	p {
	letter-spacing: 2px;
	}
	
	Combining Text Properties
	
	p {
	color: #333;
	text-align: justify;
	text-decoration: underline;
	text-transform: capitalize;
	letter-spacing: 1px;
	word-spacing: 2px;
	line-height: 1.8;
	text-indent: 20px;
	}			

Q3.b	What is responsive web design? Why is it important? **What is responsive web design? Why is it important?** **Responsive Web Design (RWD)** is an approach to building websites so that the layout and content adapt smoothly to different screen sizes and device types — phones, tablets, laptops, desktops — without needing separate sites for each. It uses flexible grids, fluid images, and CSS media queries to change styles based on viewport size, orientation, or other device features. **Why it's important** ● **Better user experience**: Visitors can read and navigate easily on any device. ● **Single codebase**: One site works across devices, simpler to maintain than separate mobile/desktop sites.	6	L2	CO2

- **SEO benefits**: Search engines (like Google) prefer mobile-friendly, responsive sites.

- **Wider reach**: Supports users on phones, tablets, desktops — increases engagement and conversions.

- **Performance**: Proper responsive design helps deliver appropriate assets (sizes) for devices, improving load times.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Responsive Example</title>
  <style>
    .container {
      display: flex;
      gap: 10px;
    }
    .box {
      flex: 1;
      padding: 20px;
      background: lightblue;
      text-align: center;
      font-size: 20px;
    }
    /* Responsive: On small screens, stack boxes vertically */
    @media (max-width: 600px) {
      .container {
        flex-direction: column;
      }
    }
  </style>
</head>
<body>
  <h2>Responsive Web Design Example</h2>
  <p>Resize the browser window to see the effect.</p>

  <div class="container">
    <div class="box">Box 1</div>
    <div class="box">Box 2</div>
    <div class="box">Box 3</div>
  </div>
</body>
</html>
```

Q3.c	Write a CSS rule to change the background color of all <h1> elements to blue. Explain the selector and property used. **1. Selector: h1** ● In CSS, a **selector** is used to target the HTML elements you want to style. ● Here, the selector is h1, which means this rule will apply to **all <h1> elements** in the webpage. ● If a page has multiple <h1> tags, this rule ensures that each one will receive the same background color. Example in HTML: <h1>Welcome to My Website</h1> <h1>About Us</h1> ● Both headings above would be styled with a blue background. **2. Property: background-color** ● In CSS, a **property** defines what aspect of the element you want to change. ● The property background-color controls the **background color** of the selected element. ● It can take values such as: ○ Color names (e.g., blue, red, green) ○ HEX codes (e.g., #0000FF for blue) ○ RGB values (e.g., rgb(0, 0, 255)) ○ HSL values (e.g., hsl(240, 100%, 50%) for blue) In this case, background-color: blue; sets the background behind the text of all <h1> headings to **blue**. <!DOCTYPE html>	7	L3	CO2

	<html>
	<head>
	<style>
	h1 {
	background-color: blue; /* CSS rule applied */
	color: white; /* Text color changed for better visibility */
	padding: 10px; /* Adds space around text */
	}
	</style>
	</head>
	<body>
	 Main Heading
	 Another Heading
	</body>
	</html>			

	OR			
Q4.a	What is Bootstrap? Mention two benefits of using it in web development. **Bootstrap** is a popular open-source front-end framework used for designing responsive and mobile-first websites. It provides pre-designed HTML, CSS, and JavaScript components such as navigation bars, buttons, forms, grids, and modals that help developers build websites quickly and efficiently. **Two benefits of using Bootstrap:** 1. **Responsive Design** – Bootstrap's grid system and ready-made classes ensure that websites automatically adjust and look good on all screen sizes (mobile, tablet, desktop). 2. **Faster Development** – With built-in templates, components, and utilities, developers can save time and effort instead of writing CSS and JavaScript from scratch.	4	L2	CO2
Q4.b	Describe how you would implement a mobile-first design using CSS **Mobile-First Design with CSS** A **mobile-first design** means creating a website that is optimized for small screens (like smartphones) first, and then gradually enhancing the layout for larger devices (like tablets and desktops) using **media queries**. **Steps to Implement**	6	L2	CO2

1. **Start with Base Styles for Mobile**

    - Write simple CSS that works well on small screens by default.
    - Use a single-column layout, larger touch-friendly buttons, and readable font sizes.

```
body {
  font-family: Arial, sans-serif;
  font-size: 16px;
  margin: 10px;
}
.container {
  display: block; /* simple stacking layout */
}
```

2. **Use Relative Units**

    - Use %, em, rem, vw, and vh instead of fixed px to ensure flexibility across devices.

3. **Apply Fluid Images and Elements**

    - Ensure images and videos don’t overflow the screen.

```
img {
  max-width: 100%;
  height: auto;
}
```

4. **Enhance Layout with Media Queries**

    - Add breakpoints for larger screens (min-width) to adjust the design step by step.

```
@media (min-width: 600px) {
  .container {
    display: flex;      /* two-column layout for tablets */
    gap: 20px;
  }
```

	}
	
	@media (min-width: 900px) {
	.container {
	max-width: 1100px; /* center content on desktops */
	margin: auto;
	}
	}
	
	5. **Progressive Enhancement**
	
	○ Instead of hiding content for mobile, start with the simplest version and **add more features/styles** for larger devices.
	
	Advantages of Mobile-First Approach
	
	● Ensures websites are usable on the most common devices (mobiles).
	
	● Improves performance by loading lightweight styles first.
	
	● Easier maintenance since larger layouts are just enhancements.			

Q4.c	Show the effect of flexible images with an example (HTML and CSS snippet). **Flexible Images in Responsive Web Design** Flexible images automatically scale to fit the width of their parent container, preventing overflow on small screens. This is done using CSS with max-width: 100% and height: auto. **Example (HTML + CSS)** <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Flexible Image Example</title> <style> img { max-width: 100%; /* Image will shrink if container is small */ height: auto; /* Keeps the aspect ratio */ border: 2px solid black;	5	L2	CO2

	}
	.container {
	width: 50%; /* Change this to see the effect */
	border: 2px solid blue;
	}
	</style>
	</head>
	<body>
	 Flexible Image Demo
	
	<img src="https://via.placeholder.com/800x400" alt="Sample Image">
	
	</body>
	</html>			

Q4.d	Write a CSS snippet using class selectors and apply it to an HTML element.
	
	CSS Snippet Using Class Selector
	
	A **class selector** in CSS is defined with a dot (.) before the class name. It is used to style one or more elements that share the same class.
	
	Example
	<!DOCTYPE html>
	<html lang="en">
	<head>
	<meta charset="UTF-8">
	<title>Class Selector Example</title>
	<style>
	.highlight {
	background-color: yellow; /* sets background */
	color: red; /* sets text color */
	font-weight: bold; /* makes text bold */
	padding: 10px; /* adds spacing */
	}
	</style>
	</head>
	<body>
	<h1 class="highlight">This is a highlighted heading</h1>
	 This paragraph is normal.
	 This paragraph is also highlighted using the same class.
	</body>
	</html>
		5	L3	CO2

	Module-3			

Q5.a	Write a JavaScript program to find the largest of three numbers using if-else and explain the logic.
	
	<!DOCTYPE html>
	<html lang="en">
	<head>
	<meta charset="UTF-8">
	<title>Largest Number</title>
	</head>
	<body>
	<script>
	// Three numbers
	let a = 25, b = 42, c = 18;
	let largest;
	
	// Using if-else to compare
	if (a >= b && a >= c) {
	largest = a;
	} else if (b >= a && b >= c) {
	largest = b;
	} else {
	largest = c;
	}
	
	// Output
	document.write("The largest number is: " + largest);
	</script>
	</body>
		10	L3	CO3

	</html>			
Q5.b	Describe various JavaScript operators with at least two examples for each type. What is an Operator? In JavaScript, an operator is a symbol that performs an operation on one or more operands, such as variables or values, and returns a result. Let us take a simple expression 4 + 5 is equal to 9. Here 4 and 5 are called operands, and + is called the operator. JavaScript supports the following types of operators. ● Arithmetic Operators ● Comparison Operators ● Logical (or Relational) Operators ● Bitwise Operators ● Assignment Operators 1.JavaScript Arithmetic Operators The JavaScript arithmetic operators are used to perform mathematical calculations such as addition, multiplication, subtraction, division, etc. on numbers. JavaScript supports the following arithmetic 2.JavaScript Comparison Operators The comparison operators in JavaScript compare two variables or values and return a boolean value, either true or false based on the comparison result. For example, we can use the comparison operators to check whether two operands are equal or not. The comparison operators are used in logical expressions. A logical expression is evaluated to either true or false. The comparison operators are binary operators as they perform operations on two operands. The operands can be numerical, string, logical, or object values.	10	L2	CO3

There are eight comparison operators in JavaScript to perform different types of comparison. Here,

we have given a table explaining each comparison operator with the example.

Operator Description Example

== Equal x == y

!= Not Equal x != y

=== Strict equality (equal value and equal type) x === y

!== Strict inequality (not equal value or not equal type) x !== y

Operator Description Example

+ (Addition) Adds two operands. x + y will give 30.

- (Subtraction) Subtracts the second operand from

the first. x - y will give -10.

* (Multiplication) Multiplies both operands. x * y will give 200.

/ (Division) Divides the numerator by the denominator.

y / x will give 2.

% (Modulus) Outputs the remainder of an

integer division. y % x will give 0

++ (Increment) Increases an integer value by one. x++ will give 11.

-- (Decrement) Decreases an integer value by one. x-- will give 9.

> Greater than x > y

< Less than x < y<

>= Greater than or Equal to x >= y

&lt= Less than or Equal to x &lt= y

3.JavaScript Logical Operators

The logical operators in JavaScript are generally used with Boolean operands and return a boolean

value. There are mainly three types on logical operators in JavaScript - && (AND), || (OR), and !

(NOT). These operators are used to control the flow of the program.

Although the logical operators are typically used with Boolean values, they can be used with any

type. For each non-boolean value, the operator converts to a boolean. The falsy values are

converted to false and truthy values to true.

The && and || operators return the value of one of the operands based on condition. So if the

operands are non-boolean, they return a non-boolean value. The ! operator always returns a Boolean

value.

The operands may be literals, variables or expressions. These are first evaluated to the boolean

equivalent before performing the logical operation.

In the below table, we have given the logical operators with its description and example. Let's

assume: x = true, y = false.

Operator Description Example

&& Logical AND (x && y) is false.

|| Logical OR (x || y) is true.

! Logical NOT !(x) is false.

JavaScript Logical AND (&&) Operator

The logical AND (&&) operator evaluates the operands from left to right. If the first operand can be

converted to false, it will return the value of the first operand, otherwise it will return the value of

the second operand.

x && y

In the above expression if x is a falsy value then it will return the value of x otherwise it will return

the value of y.

The above rule is followed for all types of operands, whether they are Boolean values, numbers or

strings, etc.

Let's first discuss Boolean operands. In general, for a set of Boolean operands, it will return true if

both operands are true else it returns false.

JavaScript Logical OR (||) Operator

The logical OR (||) operator also evaluates the operands from left to right. If the first operand can be

converted to true, it will return the value of first operand, otherwise it will return the value of the

second operand.

x || y

In the above expression if x is a truthy value then it will return the value of x otherwise it will return

the value of y.

As || is a logical operator but it can be applied to any type of operand not only boolean.

Let's first discuss Boolean operands. In general, for a set of Boolean operands, it will return false if

both operands are false else it returns true.

JavaScript Logical NOT (!) Operator

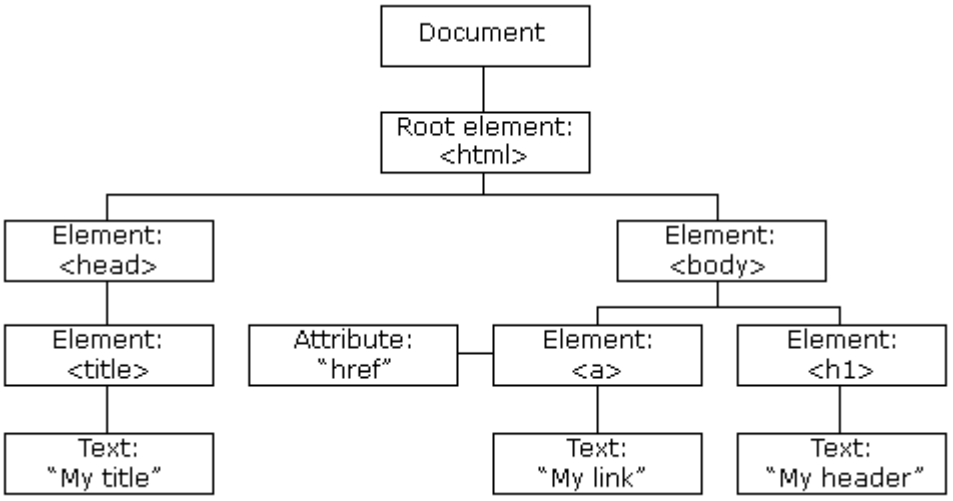
The logical NOT (!) Operator is a unary operator. It returns false if the operand can be converted to

true, otherwise it returns true.

!x

If x is truthy, the NOT (!) operator returns false. If the x is false then it returns true.

Same as Logical AND, and OR operators, this logical NOT operator can also be used with

	non-boolean operands. But it will always return a Boolean value.			
	OR			
Q6.a	What is the Document Object Model (DOM)? Explain its structure and purpose. **What is DOM?** The Document Object Model (DOM) is a programming interface for web documents. It represents the structure of a document as a tree of objects, where each object corresponds to a part of the document, such as elements, attributes, and text. JavaScript can manipulate this tree structure, allowing developers to dynamically alter the content and appearance of a webpage.**JavaScript can interact with the DOM to dynamically change content, structure, and styles on a webpage.** **DOM Structure** The HTML DOM model is constructed as a tree of Objects: The HTML DOM Tree of Objects  With the object model, JavaScript gets all the power it needs to create dynamic HTML: ● JavaScript can change all the HTML elements in the page ● JavaScript can change all the HTML attributes in the page ● JavaScript can change all the CSS styles in the page ● JavaScript can remove existing HTML elements and attributes ● JavaScript can add new HTML elements and attributes ● JavaScript can react to all existing HTML events in the page	10	L2	CO3

	● JavaScript can create new HTML events in the page **DOM Manipulation** The document object represents your web page. If you want to access any element in an HTML page, you always start with accessing the document object. Below are some examples of how you can use the document object to access and manipulate HTML. --- Finding HTML Elements **Method** — **Description** document.getElementById(*id*) — Find an element by element id document.getElementsByTagName(*name*) — Find elements by tag name document.getElementsByClassName(*name*) — Find elements by class name			
Q6.b	What is AJAX? Explain how it helps in loading data without reloading the page. **AJAX** **AJAX (Asynchronous JavaScript and XML)** is a technique in web development used to send and receive data from a server **asynchronously** (in the background) without needing to reload the entire web page. Although the name contains "XML," today AJAX commonly works with data formats like **JSON**. **How AJAX Works** 1. A browser event occurs (e.g., button click). 2. JavaScript creates an **XMLHttpRequest** (or uses fetch() API). 3. The request is sent to the server in the background. 4. The server processes the request and sends data back.	10	L3	CO3

5. JavaScript updates part of the web page with the new data **without reloading**.

Why AJAX is Useful

- **Improves User Experience** → Only part of the page updates, making websites faster and smoother.
- **Saves Bandwidth** → Only required data is transferred, not the whole page.
- **Real-Time Updates** → Used in live chat apps, stock updates, weather apps, etc.

Example: Loading Data without Reloading

```
<!DOCTYPE html>
<html>
<head>
 <title>AJAX Example</title>
 <script>
  function loadData() {
   // Create an XMLHttpRequest object
   let xhr = new XMLHttpRequest();

   // Define what happens when data is ready
   xhr.onreadystatechange = function () {
    if (xhr.readyState == 4 && xhr.status == 200) {
     document.getElementById("data").innerHTML = xhr.responseText;
    }
   };

   // Send request to server file (sample.txt)
   xhr.open("GET", "sample.txt", true);
   xhr.send();
  }
 </script>
</head>
<body>
 <h2>AJAX Demo</h2>
 <button onclick="loadData()">Load Data</button>
 <div id="data">Content will load here...</div>
</body>
</html>
```

	Module-4			
Q7.a	List any three popular front-end frameworks and describe their key features. **1. React.js (Developed by Facebook)** React is a **JavaScript library** often considered a framework because of its wide ecosystem. It is mainly used for building **user interfaces**. **Key Features:** ● **Component-Based Architecture**: The UI is divided into reusable components, making development easier and modular. ● **Virtual DOM**: React updates only the changed parts of the DOM, improving speed and performance. ● **Unidirectional Data Flow**: Data flows in a single direction, which makes debugging and state management simpler. ● **Strong Ecosystem**: React works well with libraries like Redux (state management) and React Router (routing). 👉 Example use: Instagram's web app uses React for smooth rendering and updates. **2. Angular (Developed by Google)** Angular is a **full-fledged front-end framework** based on TypeScript, suitable for building **large-scale enterprise web applications**. **Key Features:** ● **Two-Way Data Binding**: Automatically syncs data between model (logic) and view (UI). ● **Dependency Injection (DI)**: Manages object creation and sharing, making code more maintainable. ● **Directives**: Special HTML attributes (like *ngIf, *ngFor) that add dynamic behavior to web pages. ● **Built-in Tools**: Provides routing, form validation, and HTTP services out-of-the-box.	6	L2	CO4

	👉 Example use: Google applications like Gmail and Google Cloud Console use Angular. **3. Vue.js** Vue is a **progressive framework** that is lightweight yet powerful, making it popular for both small and large projects. **Key Features:** ● **Reactive Data Binding**: Like Angular, Vue supports two-way data binding, making UI updates automatic. ● **Easy Learning Curve**: Simple syntax, making it beginner-friendly compared to Angular. ● **Component-Based**: Supports reusable, modular components like React. ● **Integration Flexibility**: Vue can be integrated into existing projects gradually (progressive nature).			
Q7.b	Explain the concept of scope in AngularJS with a simple example. **oncept of Scope in AngularJS** In **AngularJS**, **Scope** is a special JavaScript object that **binds the controller and the view (HTML)**. ● It acts as a **glue** between the **Model (data)** and the **View (UI)**. ● The $scope object stores the application data (variables, functions) and makes them available to the view. ● Whenever data inside the scope changes, the view automatically updates (**two-way data binding**). ● Each AngularJS application has a **root scope** ($rootScope) and can have **multiple child scopes** inside controllers and directives. **Simple Example** **HTML + AngularJS Code:** <!DOCTYPE html> <html ng-app="myApp">	7	L3	CO4

	<head>
	<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
	</head>
	<body ng-controller="myController">
	
	 Scope Example in AngularJS
	
	<!-- Binding scope variable to input -->
	Enter your name: <input type="text" ng-model="name">
	
	<!-- Displaying value from scope -->
	 Hello, {{name}}!
	
	</body>
	
	<script>
	// Define AngularJS app
	var app = angular.module("myApp", []);
	
	// Define controller with $scope
	app.controller("myController", function($scope) {
	$scope.name = "Daya"; // initial value stored in scope
	});
	</script>
	</html>			

Q7.c	What are AngularJS expressions? How are they different from JavaScript expressions? **AngularJS Expressions** ● AngularJS expressions are written inside **double curly braces {{ }}**. ● They are used to **bind data** between the **model and view**. ● These expressions can contain variables, operators, and function calls defined in the AngularJS **scope**. ● AngularJS evaluates the expression against the **$scope object**, not the global window object.	7	L2	CO4

	Example:
	
	 Total: {{ price * quantity }}
	
	● If $scope.price = 50 and $scope.quantity = 3, output will be **150**.
	
	JavaScript Expressions
	
	● JavaScript expressions are written in **plain JS code** and run inside the browser's **global scope** (window).
	
	● They can include variables, operators, function calls, objects, arrays, etc.
	
	● Must be explicitly written inside <script> tags or external .js files.
	
	Example:
	
	var price = 50;
	var quantity = 3;
	
	● document.write("Total: " + (price * quantity));			

	OR			
Q8.a	Compare React, Angular, and Vue based on their structure and features. **Comparison of React, Angular, and Vue** **1. React.js** ● **Structure**: A **JavaScript library** (not a full framework) developed by Facebook. Uses a **component-based architecture**. ● **Features**: ○ **Virtual DOM** for fast UI updates. ○ **Unidirectional data flow** (one-way binding). ○ Works with external libraries like Redux for state management. ○ Flexible and lightweight but needs integration with other tools for routing and forms.	8	L2	CO4

	2. Angular ● **Structure**: A **full-fledged framework** developed by Google, written in TypeScript. ● **Features**: ○ **Two-way data binding** for automatic UI updates. ○ **Dependency Injection (DI)** for managing services and components. ○ Provides built-in tools like **routing, form validation, HTTP services**. ○ Strongly opinionated and suited for large-scale enterprise apps. **3. Vue.js** ● **Structure**: A **progressive framework** created by Evan You. It is lightweight, combining the best ideas of Angular and React. ● **Features**: ○ **Reactive two-way data binding** like Angular. ○ **Component-based architecture** like React. ○ Easy to integrate with existing projects. ○ Simple learning curve, suitable for beginners.			
Q8.b	List the steps to create a simple AngularJS application. **Steps to Create a Simple AngularJS Application** Creating an AngularJS application involves **initializing the app, creating a controller, and binding data to the view**. **Step 1: Include AngularJS Library** ● First, include the AngularJS library in your HTML file. ● You can use a **CDN link**:	5	L3	CO4

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

Step 2: Initialize AngularJS Application

- Use the **ng-app directive** in the <html> or <body> tag to define the AngularJS application.

<body ng-app="myApp">

Step 3: Create a Controller

- Use the **ng-controller directive** to attach a controller to a part of your page.
- Define the controller in JavaScript and attach it to the AngularJS app.

```
var app = angular.module("myApp", []);

app.controller("myController", function($scope) {
   $scope.message = "Hello, AngularJS!";
});
```

Step 4: Bind Data to the View

- Use **AngularJS expressions {{ }}** to display variables from $scope in HTML.

{{ message }}

- The data from the controller is automatically shown in the view.

Step 5: Add Interactivity (Optional)

- Use **directives** like ng-model for input binding and ng-click for events.

```
Enter your name: <input type="text" ng-model="name">
<p>Hello, {{ name }}!</p>
```

Step 6: Run the Application

- Open the HTML file in a **browser**.
- AngularJS automatically compiles the view and updates the DOM with bound data.

Final Simple HTML Example

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
                                                                    <scri
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body ng-controller="myController">

  <h2>Simple AngularJS App</h2>
  <p>{{ message }}</p>

  Enter your name: <input type="text" ng-model="name">
  <p>Hello, {{ name }}!</p>

  <script>
    var app = angular.module("myApp", []);
    app.controller("myController", function($scope) {
        $scope.message = "Hello, AngularJS!";
    });
  </script>

</body>
</html>
```

Q8.c | Compare Angular and React in terms of architecture and usage. | 7 | L2 | CO4

Comparison: Angular vs React

1. Architecture

Aspect	Angular	React
Type	Full-fledged front-end **framework**	JavaScript **library** for building UI

Structure	**MVC / MVVM architectu** (Model-View-Controller Model-View-ViewModel). Provid **everything out-of-the-box** includir routing, forms, HTTP services.	**Component-based architecture**. Focused only on **UI components**; needs external libraries for routing, state management, and HTTP requests.
Data Binding	**Two-way data binding** – changes the UI automatically update the mod and vice versa.	**One-way data binding** – data flows in a single direction; state management libraries like Redux are used for complex apps.
DOM	Real DOM with **change detecti** mechanism	**Virtual DOM** for faster and efficient updates

2. Usage / Learning Curve

Aspect	Angular	React
Learning Curve	Steep; requires knowledge TypeScript, dependency injectio decorators, and Angular-specif concepts.	Moderate; easier for developers familiar with JavaScript and JSX.
Best For	Large-scale **enterpri applications** with compl architecture.	Flexible, **UI-focused applications**, single-page apps, and projects requiring custom integrations.
Ecosystem	Provides **everything built-** (routing, forms, services).	Lightweight; relies on external libraries for routing, state management, and advanced features.
Performance	Slightly slower for heavy updat due to real DOM, but optimized b change detection.	Faster updates due to **Virtual DOM**. Ideal for apps with dynamic, frequently changing UIs.

	Module-5			
Q9.a	Differentiate between SQL and NoSQL databases with examples. **SQL vs NoSQL Databases**	7	L2	CO5

1. Definition

- **SQL Databases**: Relational databases that store data in **tables with rows and columns**. Use **Structured Query Language (SQL)** to manage and query data.
- **NoSQL Databases**: Non-relational databases that store data in **flexible formats** such as key-value pairs, documents, graphs, or wide-columns. They are **schema-less** or have dynamic schemas.

2. Comparison Table

Feature	SQL Databases	NoSQL Databases
Data Model	Relational (tables, rows, columns)	Non-relational (key-value, document, graph, column)
Schema	Fixed schema (predefined structure)	Dynamic schema (flexible, can change over time)
Query Language	SQL (Structured Query Language)	Varies by type; e.g., MongoDB uses JSON-style queries
Scalability	Vertical scaling (adding more CPU/RAM to single server)	Horizontal scaling (adding more servers to distribute data)
Transactions	Supports ACID (Atomicity, Consistency, Isolation, Durability)	Many support BASE (Basically Available, Soft state, Eventually consistent)
Examples	MySQL, PostgreSQL, Oracle, SQL Server	MongoDB, Cassandra, Redis, CouchDB
Use Case	Applications needing structured data and complex queries (banking, ERP)	Applications with large-scale, unstructured, or rapidly changing data (social media, IoT, big data)

3. Key Differences

1. **Structure:** SQL uses structured tables, NoSQL is schema-less.
2. **Flexibility:** SQL requires predefined columns; NoSQL can store different data formats in the same database.
3. **Scaling:** SQL is vertically scalable, NoSQL is horizontally scalable.

	4. **Transactions:** SQL ensures strong consistency (ACID); NoSQL often prioritizes availability and partition tolerance (BASE). 5. **Examples:** MySQL (SQL), MongoDB (NoSQL).			
Q9.b	Mention any two server-side languages and list the advantages of each. **Two Server-Side Languages and Their Advantages** **1. PHP (Hypertext Preprocessor)** **Advantages:** 1. **Open Source and Free** – PHP is free to use, which reduces development cost. 2. **Easy Integration with Databases** – Works seamlessly with databases like MySQL, PostgreSQL, and SQLite. 3. **Cross-Platform** – Runs on Windows, Linux, and macOS servers. 4. **Large Community Support** – Many libraries, frameworks (like Laravel, CodeIgniter) are available. **Example Use:** WordPress, Facebook (initially), e-commerce websites. **2. Python** **Advantages:** 1. **Simple and Readable Syntax** – Easy to learn and maintain, which speeds up development. 2. **Wide Range of Frameworks** – Frameworks like Django and Flask help build secure, scalable web applications quickly. 3. **Supports Multiple Paradigms** – Object-oriented, procedural, and functional programming. 4. **Large Community and Libraries** – Many libraries for web development, data analysis, AI, and more.	7	L2	CO5

Q9.c	Define authentication and authorization in web applications. **Authentication and Authorization in Web Applications** **1. Authentication** ● **Definition:** Authentication is the process of **verifying the identity of a user** or system before granting access to an application. ● It ensures that the person trying to access the application is **who they claim to be**. ● **Common Methods:** ○ Username and password ○ OTP (One-Time Password) ○ Biometric verification (fingerprint, face recognition) ○ OAuth / Social logins (Google, Facebook) **Example:** When a user enters a username and password on a login page, the system checks if the credentials are correct. If yes, the user is authenticated. **2. Authorization** ● **Definition:** Authorization is the process of **determining what an authenticated user is allowed to do** within the application. ● It ensures that users can **only access resources or perform actions they are permitted to**. ● **Common Methods:** ○ Role-based access control (RBAC) – Admin, User, Guest ○ Permission levels – Read, Write, Delete **Example:** A regular user can view their profile but cannot access the admin dashboard. The system checks the user's role and authorizes access accordingly.	6	L3	CO5
	OR			

Q10.a	Describe the concept of full-stack development with a real-time example. **Full-Stack Development** **Definition:** Full-stack development is the practice of building both the **front-end (client-side)** and **back-end (server-side)** parts of a web application. A **full-stack developer** is capable of handling **everything from designing the user interface to managing databases and server logic**. ● **Front-End (Client-Side):** Responsible for what users see and interact with. Uses technologies like **HTML, CSS, JavaScript**, and frameworks/libraries like **React, Angular, Vue**. ● **Back-End (Server-Side):** Handles the server, application logic, and database interactions. Uses languages like **Node.js, Python, PHP, Java** and databases like **MySQL, MongoDB**. ● **Full Stack:** Combines front-end + back-end + database knowledge, enabling development of **complete applications**. **Key Responsibilities of a Full-Stack Developer** 1. Designing and developing responsive user interfaces. 2. Implementing server-side logic and APIs. 3. Managing databases and data storage. 4. Ensuring security, authentication, and authorization. 5. Deploying applications to cloud servers or hosting platforms. **Real-Time Example: Online Food Delivery Application** **Scenario:** Building a food delivery app like **Zomato or Swiggy**. ● **Front-End (React/Angular/Vue):** ○ User can browse restaurants, view menus, add items to the cart, and place an order. ○ Implements **responsive design** for mobile and desktop users.	10	L3	CO5

	● **Back-End (Node.js/PHP/Python):** ○ Handles order processing, user authentication, payment processing, and sending notifications. ○ Exposes **APIs** to the front-end for fetching restaurant data and submitting orders. ● **Database (MySQL/MongoDB):** ○ Stores user information, restaurant menus, orders, and payment history. ● **Full-Stack Role:** ○ The developer connects the front-end UI with the back-end APIs and database, ensuring smooth interaction between user actions and server responses.			
Q10.b	How would you deploy a web application on a cloud hosting platform? **Steps to Deploy a Web Application on a Cloud Hosting Platform** **1. Choose a Cloud Hosting Platform** ● Popular platforms: **AWS (Amazon Web Services), Google Cloud Platform (GCP), Microsoft Azure, Heroku, Netlify, Vercel**. ● Decide based on your **project requirements**, like scalability, ease of use, or cost. **2. Prepare Your Web Application** ● Ensure your application is **production-ready**: ○ Front-end built (HTML, CSS, JS, or frameworks like React/Angular/Vue). ○ Back-end configured (Node.js, Python, PHP, etc.) with database connections. ○ Configuration files updated (like .env for environment variables).	10	L3	CO5

3. Set Up a Cloud Instance or Hosting Service

- For **IaaS** (Infrastructure as a Service, e.g., AWS EC2):
    - Create a virtual server instance.
    - Install necessary software (web server like Apache/Nginx, language runtime).
- For **PaaS** (Platform as a Service, e.g., Heroku, Netlify, Vercel):
    - Simply link your repository (GitHub/GitLab).
    - Configure build commands and deployment settings.

4. Upload or Connect Your Application

- **Option 1: Git Deployment**
    - Push your code to a Git repository.
    - Connect the repository to the cloud platform (Heroku, Netlify, Vercel).
- **Option 2: File Upload / FTP**
    - Upload files to the cloud server using FTP/SFTP.
    - Place files in the web server directory (e.g., /var/www/html for Apache).

5. Configure the Database (if required)

- Set up a database instance (MySQL, PostgreSQL, MongoDB).
- Update your application configuration with **database URL, username, and password**.
- Ensure database is accessible from your deployed app.

6. Set Up Environment Variables

- Configure variables like API_KEY, DB_URL, or PORT without hardcoding them.
- This keeps sensitive data secure and allows easy configuration across environments.

7. Start the Application

- For back-end apps: Run the server (e.g., node app.js) or let PaaS handle it automatically.
- Ensure the server is **listening on the correct port** and reachable via the internet.

8. Test the Deployment

- Access the application using the cloud URL or custom domain.
- Verify all features work: UI, database, APIs, and authentication.

9. Set Up a Domain and SSL (Optional but Recommended)

- Register a domain name.
- Configure DNS to point to your cloud application.
- Enable SSL (HTTPS) for secure connections.

Example Scenario: Deploying a Node.js App on Heroku

1. Create a Heroku account.
2. Install Heroku CLI.

Navigate to your app folder and run:

```
 git init
heroku create my-app
git add .
git commit -m "Initial commit"
```

	3. git push heroku main			