


CMR INSTITUTE OF TECHNOLOGY	USN <table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>											

VTU Theory Examination April. 2025				
Sub:	Web Technologies	Code:	MMC105	
Answer Key		Marks	OBE	
			CO	RB T
Q1(a)	<p>Define HTTP. Explain the different phases of HTTP.</p> <ul style="list-style-type: none"> HTTP What is HTTP? HTTP (Hypertext Transfer Protocol) is the foundational communication protocol used for transferring data over the web. It enables the exchange of information between a client (such as a web browser) and a server (where the website or resource is hosted). HTTP is a stateless protocol, meaning each request is independent, and no connection information is retained between requests. Features of HTTP: <ol style="list-style-type: none"> Request-Response Model: The client sends an HTTP request to the server. The server processes the request and sends back an HTTP response. Stateless: Each HTTP request is treated as an independent interaction, with no memory of previous requests. Human-Readable: HTTP messages are plain text, making them easy to read and debug. Flexible: HTTP can transmit various types of data, such as HTML, images, videos, JSON, etc. <p>How HTTP Works:</p> <ol style="list-style-type: none"> A user enters a URL (e.g., http://www.example.com) into a web browser. The browser sends an HTTP request to the web server hosting the domain example.com. The web server processes the request and returns an HTTP response, which contains the requested resource (e.g., a web page). The browser renders the received data into a readable format for the user. <p>HTTP Methods: HTTP defines several methods (also called verbs) for different types of operations:</p> <ol style="list-style-type: none"> GET: HOST:/test/demo_form.php?name1=value1&name2=value2 GET requests can be cached GET requests remain in the browser history 	10	L2	CO1

	<p>GET requests can be bookmarked GET requests should never be used when dealing with sensitive data GET requests have length restrictions GET requests are only used to request data (not modify)</p> <p>2. POST: Host: w3schools.com</p> <ul style="list-style-type: none"> o POST requests are never cached o POST requests do not remain in the browser history o POST requests cannot be bookmarked o POST requests have no restrictions on data length <p>3. PUT: PUT is used to send data to a server to create/update a resource. The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.</p> <p>4. DELETE: Deletes a resource on the server.</p> <p>5. HEAD: HEAD is almost identical to GET, but without the response body. In other words, if GET /users returns a list of users, then HEAD /users will make the same request but will not return the list of users. A HEAD request is useful for checking what a GET request will return before actually making a GET request - a HEAD request can read the Content-Length header to check the size of the file, without actually downloading the file.</p> <p>6. OPTIONS: Describes the communication options for the resource.</p> <p>7. PATCH: Partially updates a resource.</p> <p>HTTP Status Codes: Servers use status codes in HTTP responses to indicate the result of a request: 2xx (Success): The request was successful (e.g., 200 OK). 3xx (Redirection): The client is redirected to another location (e.g., 301 Moved Permanently). 4xx (Client Error): There was an error with the client request (e.g., 404 Not Found). 5xx (Server Error): The server failed to process the request (e.g., 500 Internal Server Error).</p>			
Q1(b)	<p>Discuss the basic structure of XHTML documents. Also explain the rules to be followed to make use of HTML elements in XHTML documents.</p> <p>Basic Structure of XHTML Documents XHTML (Extensible Hypertext Markup Language) is a reformulation of HTML as an XML application. It combines the flexibility of HTML with the strictness of XML. Because of its XML roots, XHTML documents must follow stricter syntax rules than traditional HTML. Here's the basic structure of an XHTML document:</p> <pre><?xml version="1.0" encoding="UTF-8"?></pre>	10	L2	CO1

<pre> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> <html xmlns="http://www.w3.org/1999/xhtml"> <head> <title>Example XHTML Page</title> </head> <body> <h1>Hello, XHTML!</h1> <p>This is a sample XHTML document.</p> </body> </html> </pre> <p>Breakdown of Structure:</p> <ol style="list-style-type: none"> 1. XML Declaration: <code><?xml version="1.0" encoding="UTF-8"?></code> – Optional but recommended, specifies XML version and character encoding. 2. DOCTYPE Declaration: Defines the document type and DTD (Document Type Definition) used. 3. Root Element <html>: <ul style="list-style-type: none"> ○ Must include the XML namespace: <code>xmlns="http://www.w3.org/1999/xhtml"</code> 4. Head Section: Contains metadata, including <code><title></code>. 5. Body Section: Contains visible content like headings, paragraphs, etc. <p>Rules for Using HTML Elements in XHTML Documents Because XHTML is based on XML, it must conform to strict syntax rules. The key rules include:</p> <ol style="list-style-type: none"> 1. All tags must be properly closed: <ul style="list-style-type: none"> ○ Example: <code>
</code>, <code><hr /></code>, <code></code> ○ Even empty elements must be closed. 2. All tags must be properly nested: <ul style="list-style-type: none"> ○ Correct: <code><p>Bold text</p></code> ○ Incorrect: <code><p>Bold text</p></code> 3. All tag names and attributes must be in lowercase: <ul style="list-style-type: none"> ○ XHTML is case-sensitive, unlike HTML. ○ Use <code></code> not <code></code> 4. Attribute values must be quoted: <ul style="list-style-type: none"> ○ Correct: <code><input type="text" value="Name" /></code> ○ Incorrect: <code><input type=text value=Name></code> 5. Documents must have a DOCTYPE declaration: <ul style="list-style-type: none"> ○ This helps browsers render the document correctly and validates the structure. 6. The root element must include the XHTML namespace: <ul style="list-style-type: none"> ○ Example: <code><html xmlns="http://www.w3.org/1999/xhtml"></code> 7. Avoid deprecated elements and attributes: <ul style="list-style-type: none"> ○ Use CSS for styling instead of older attributes like <code>align</code>, <code>bicolor</code>, etc. 			
---	--	--	--

Q2(a)	Briefly explain the following: 1. URL 2. MIME 3. Web server 4. Web Browser ● <u>URL</u>	10	L2	CO1
-------	---	----	----	-----

A URL (Uniform Resource Locator) is the unique address used to identify and access resources on the internet, such as web pages, files, or images. It specifies where a resource is located and how to retrieve it.

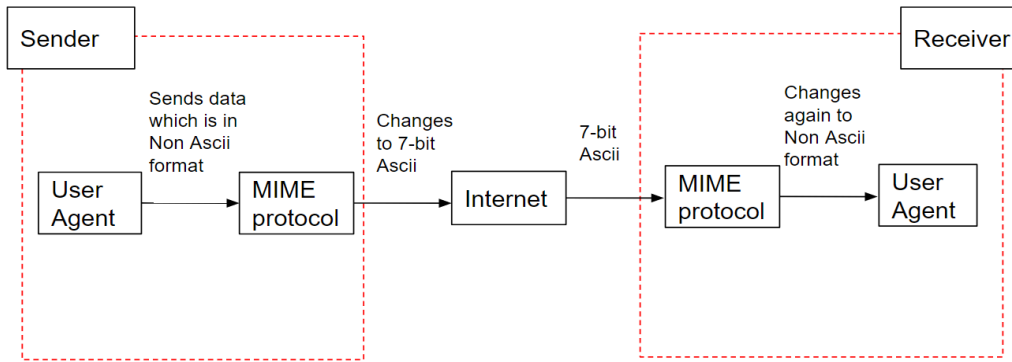
Components of a URL

1. Protocol:
Indicates the communication protocol to use (e.g., HTTP, HTTPS, FTP).
2. Domain:
Identifies the server hosting the requested resource.
Can be a human-readable name (e.g., example.com) or an IP address (e.g., 192.168.1.1).
Case insensitive.
3. Port (Optional):
Specifies the port number to connect to on the server.
Default ports are determined by the protocol:
HTTP: Port 80.
HTTPS: Port 443.
Non-default ports can be specified using a colon after the domain, e.g., http://example.com:888/.
4. Path (Optional):
Represents the location of a file or directory on the server.
Follows the domain, e.g., http://example.com/**files/image.jpg**.
Case-sensitive on most servers (except some Windows-based servers).
If not specified, the server serves the default file (e.g., index.html or default.html).
5. Query String (Optional):
Provides key-value pairs for additional information, often from user input or form submissions.
Begins with a ? symbol, with key-value pairs separated by &.
Example:
<http://example.com/page?username=john&password=abc123>.
6. Fragment (Optional):
Points to a specific part of the resource, typically within a webpage.
Starts with a # symbol.
Example: http://example.com/page#section1 directs the browser to the section1 anchor within the page.

- **MIME**

What is MIME?

MIME stands for Multipurpose Internet Mail Extensions, a standard that extends the format of email to support text in different character sets, attachments such as images, audio, video, and application files, and other multimedia formats. Although originally developed for email, MIME types are now widely used in the context of the Web, where they describe the nature and format of a file or data.



Key Features of MIME

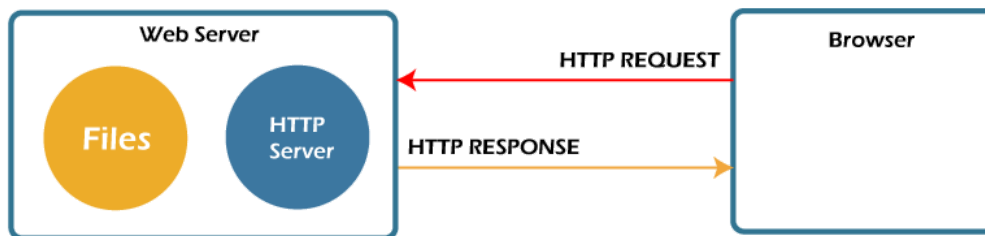
1. **Content Description:**
Specifies the type of data being sent.
Examples: Text, image, video, audio, etc.
2. **Encodings:**
Allows non-text data to be encoded in text-based formats for transmission (e.g., Base64).
3. **Multipart Messages:**
Supports messages with multiple parts (e.g., an email with both text and an attachment).
4. **Cross-Application Usage:**
Used by web browsers, servers, and email clients to handle and interpret file formats correctly.

• **web servers**

What is a Web Server?

A web server is a computer system or software application that serves content to users over the internet. It processes requests made through the Hypertext Transfer Protocol (HTTP) or its secure version, HTTPS, and delivers the requested resources, such as web pages, images, or files.

Web Servers



How a Web Server Works:

1. **Request Handling:**
A user types a URL in their browser or clicks a link.
The browser sends an HTTP request to the web server.
2. **Response Generation:**
The web server processes the request.
If the resource exists, it sends back the content (e.g., HTML, CSS, JavaScript files).
If not, the server returns an error code like 404 Not Found.
3. **Browser Rendering:**
The browser displays the returned data to the user.

Web Servers and the LAMP Stack: Overview

A web server is essentially a computer that responds to HTTP requests and delivers web content. It can range from a simple personal computer (like Tim Berners-Lee's first web server) to powerful servers in large-scale web farms.

Key Components of a Web Server:

1. **Operating Systems (OS):**
Commonly used OS: Linux (preferred for its uptime, lower memory usage, and remote management).

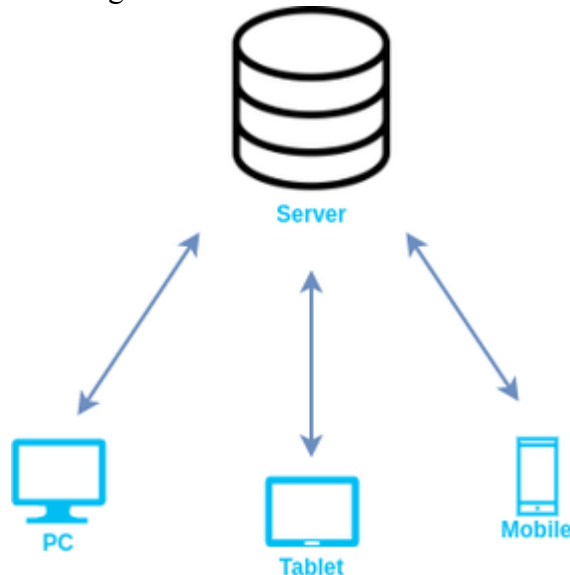
- Windows is also widely used, especially in enterprises adopting Microsoft tools.
2. Web Server Software:
 Apache: Open-source, widely used, supports Linux, Windows, and Mac.
 IIS (Internet Information Services): Microsoft's web server software, tightly integrated with the .NET framework.
 3. Database Software:
 For dynamic websites, databases are essential.
 Common open-source options: MySQL, SQLite.
 Proprietary choices: Microsoft SQL Server, Oracle, IBM DB2.
 4. Scripting/Server-Side Software:
 LAMP stack often uses PHP, but other options include Python, Ruby on Rails, or ASP.NET.
 PHP is popular for its ease of use, widespread support, and compatibility with HTTP.

The LAMP Stack:

- Linux (Operating System)
- Apache (Web Server)
- MySQL (Database Management System)
- PHP (Server-Side Scripting Language)

• **What is a Web Browser?**

A web browser is a software application that facilitates access to the World Wide Web (WWW) by acting as an intermediary between the client (user) and the server. It enables users to request web documents and services from servers, interprets the received data (usually in HTML), and renders it as a user-friendly web page containing text, images, links, and interactive elements. Common web browsers include Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.



History of Web Browsers

1. WorldWideWeb (1990):
 Invented by Tim Berners-Lee; later renamed Nexus.
 First web browser and editor.
2. Netscape Navigator (1994):
 An advanced version of Mosaic, developed by Marc Andreessen.
 Played a major role in the browser wars of the 1990s.
3. Internet Explorer (1995):
 Launched by Microsoft as the default browser for Windows OS.
 Dominated the market for years.
4. Modern Browsers:
 Mozilla Firefox, Google Chrome, Safari, Opera, and others followed,
 each offering unique features like speed, privacy, and integration.

How Does a Web Browser Work?

A web browser operates using the client-server model:

1. Client Request.
2. Server Response:

The server processes the request and sends back the required resource, typically in the form of HTML, CSS, JavaScript, and media files.

3. Rendering:

The browser interprets the received data, converts it into a graphical format, and displays the content on the screen.

4. Cookies:

Small files stored by the browser to retain user preferences, session data, and browsing patterns. Used by websites for personalization and targeted advertising.

Q2(b)

Explain the following tags with examples.

1. Heading tag
2. Hypertext link tag
3. Image tag
4. Progress tag

1. Heading Tag (<h1> to <h6>)

The heading tags are used to define headings in a web page. There are six levels of headings:

- <h1> is the largest and most important

- <h6> is the smallest

<h1>Main Title</h1>

<h2>Subheading</h2>

<h3>Section Title</h3>

<h4>Subsection</h4>

<h5>Minor Heading</h5>

<h6>Smallest Heading</h6>

2. Hypertext Link Tag (<a>)

The <a> tag defines a hyperlink, which is used to link from one page to another.

Syntax:

Link Text

Example:

Visit Example

href specifies the destination URL.

Clicking the text "Visit Example" will navigate to the specified URL.

3. Image Tag ()

The tag is used to embed images in a webpage. It is a self-closing tag.

Syntax:

Example:

- src: Path to the image file.
- alt: Alternative text for accessibility or if image fails to load.
- width and height: Optional attributes to specify image dimensions.

4. Progress Tag (<progress>)

The <progress> tag represents the completion progress of a task. It is useful for showing loading bars or task completion percentages.

Syntax:

10

L2

CO1

	<p><progress value="current" max="maximum">Fallback text</progress></p> <p>Example:</p> <p><label for="progress">Downloading:</label></p> <p><progress id="progress" value="70" max="100">70%</progress></p> <p>value: Current progress.</p> <p>max: Maximum value (typically 100).</p> <p>The progress bar will appear 70% filled in this case.</p>			
Q3(a)	<p>Discuss on the different ways of including CSS style information to a HTML document.</p> <p>There are three main ways to include CSS (Cascading Style Sheets) in an HTML document to style its elements:</p> <p>1. Inline CSS</p> <ul style="list-style-type: none"> • CSS is written directly within an HTML element using the style attribute. • Best used for quick, one-time styling. • Not recommended for larger projects due to poor maintainability. <p>Example:</p> <p><p style="color: blue; font-size: 16px;">This is a blue paragraph.</p></p> <p>2. Internal CSS (Embedded CSS)</p> <ul style="list-style-type: none"> • CSS is placed inside a <style> tag within the <head> section of the HTML document. • Useful for applying styles to a single HTML document. <p>Example:</p> <pre> <!DOCTYPE html> <html> <head> <style> body { background-color: #f2f2f2; } h1 { color: darkgreen; } p { font-family: Arial; } </style> </head> </pre>	10	L2	CO2


```

<body>
<h1>Welcome</h1>
<p>This is an example using internal CSS.</p>
</body>
</html>

```

3. External CSS

- CSS is written in a separate .css file and linked to the HTML using the <link> tag.
- Best for styling multiple pages consistently and efficiently.

CSS File (styles.css):

```

body {
  background-color: light yellow;
}
h1 {
  color: navy;
}

```

HTML File:

```

<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="styles.css" />
</head>
<body>
  <h1>External CSS Example</h1>
</body>
</html>

```

Q3(b)

Name any five CSS selectors and explain their uses with a suitable example.

1. Universal Selector (*)

- **Use:** Applies styles to **all elements** on the page.
- **Syntax:** * { property: value; }

Example:

```

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

```

This removes default margin and padding from all elements.

2. Element Selector (Tag Selector)

- **Use:** Targets all HTML elements of a specific type.

10

L3

CO2

	<ul style="list-style-type: none"> ● Syntax: elementName { property: value; } <p>Example:</p> <pre>p { color: green; font-size: 16px; }</pre> <p>Styles all <p> (paragraph) elements with green text.</p> <p>3. Class Selector (.)</p> <ul style="list-style-type: none"> ● Use: Targets all elements with a specific class attribute. ● Syntax: .className { property: value; } <p>Example:</p> <pre>.highlight { background-color: yellow; }</pre> <p><p class="highlight">This paragraph is highlighted.</p></p> <p>4. ID Selector (#)</p> <ul style="list-style-type: none"> ● Use: Targets a single element with a specific id. ● Syntax: #idName { property: value; } <p>Example:</p> <pre>#header { text-align: center; font-size: 24px; }</pre> <p><h1 id="header">Welcome!</h1></p>			
Q4(a)	<p>Explain the various ways of creating arrays in javascript. Mention any 5 array methods and Explain their use.</p> <p>Ways of Creating Arrays in JavaScript</p> <p>JavaScript provides several ways to create arrays. Here are the main methods:</p> <p>1. Using Array Literals (Most Common Way)</p> <pre>let fruits = ["apple", "banana", "cherry"];</pre> <p>2. Using the Array Constructor</p> <pre>let colors = new Array("red", "green", "blue");</pre> <p>Note: new Array(3) creates an array with 3 empty slots (not actual values).</p>	10	L2	CO2

3. Using Array.of()

```
let numbers = Array.of(1, 2, 3, 4);
```

Creates a new array with the given elements.

Useful Array Methods with Explanation

1. push()

- **Use:** Adds one or more elements to the **end** of an array.

```
let fruits = ["apple", "banana"];
```

```
fruits.push("mango");
```

```
console.log(fruits); // ["apple", "banana", "mango"]
```

2. pop()

- **Use:** Removes the **last** element from an array and returns it.

```
let nums = [1, 2, 3];
```

```
let last = nums.pop();
```

```
console.log(last); // 3
```

```
console.log(nums); // [1, 2]
```

3. shift()

- **Use:** Removes the **first** element from an array.

```
let names = ["John", "Jane", "Jim"];
```

```
names.shift();
```

```
console.log(names); // ["Jane", "Jim"]
```

4. unshift()

- **Use:** Adds one or more elements to the **beginning** of an array.

```
let colors = ["blue", "green"];
```

```
colors.unshift("red");
```

```
console.log(colors); // ["red", "blue", "green"]
```

5. forEach()

- **Use:** Executes a function for **each element** in the array.

	<pre>let numbers = [1, 2, 3]; numbers.forEach(function(num) { console.log(num * 2); }); // Output: 2, 4, 6</pre>			
Q4(b)	<p>Write a Javascript program that accepts.</p> <ol style="list-style-type: none"> 1. Input: A number n output: The first n Fibonacci numbers 2. Input: A number n output: A table of numbers from 1 to n and their squares <p>1.The first n Fibonacci numbers</p> <pre>function generateFibonacci(n) { let fibSeries = []; for (let i = 0; i < n; i++) { if (i === 0) { fibSeries.push(0); } else if (i === 1) { fibSeries.push(1); } else { fibSeries.push(fibSeries[i - 1] + fibSeries[i - 2]); } } return fibSeries; }</pre> <p>// Example usage:</p> <pre>let n = 10; // You can change this value let result = generateFibonacci(n); console.log(`The first \${n} Fibonacci numbers are:`); console.log(result.join(", "));</pre> <p>2. A table of numbers from 1 to n and their squares</p> <pre><!DOCTYPE html> <html> <head> <title>Squares Table</title> </head> <body> <h3>Enter a number:</h3> <input type="number" id="numInput" /> <button onclick="generateTable()">Generate Table</button> <h3>Number-Square Table:</h3> <div id="outputTable"></div> <script> function generateTable() { const n = parseInt(document.getElementById("numInput").value); if (isNaN(n) n <= 0) { alert("Please enter a positive number."); return; } } </script></pre>	10	L3	CO2

```

    }

    let html = "<table border='1'
cellpadding='5'><tr><th>Number</th><th>Square</th></tr>";
    for (let i = 1; i <= n; i++) {
        html += '<tr><td>${i}</td><td>${i * i}</td></tr>';
    }
    html += "</table>";
    document.getElementById("outputTable").innerHTML = html;
}
</script>
</body>
</html>

```

Q5(a) Explain Document object model(DOM) with examples.

Document Object Model (DOM) in JavaScript

The **Document Object Model (DOM)** is a programming interface provided by the browser that represents an HTML or XML document as a **tree structure**. Each element, attribute, and piece of text in the document becomes a **node** in this tree.

♦ Why DOM is Important

- Allows JavaScript to access and modify the content, structure, and style of a webpage **dynamically**.
- Enables **interaction** (e.g., responding to user input, updating content, changing styles, etc.).

DOM Tree Example for a Simple HTML Page

```

<!DOCTYPE html>
<html>
  <head>
    <title>Sample Page</title>
  </head>
  <body>
    <h1>Hello, DOM!</h1>
    <p>This is a paragraph.</p>
  </body>
</html>

```

DOM Tree Representation:

```

Document
├── html
│   ├── head
│   │   └── title
│   └── body
│       ├── h1
│       └── p

```

10

L2

CO3

Accessing DOM Elements in JavaScript

```
<body>
  <p id="demo">Hello!</p>
  <script>
    // Access the paragraph using getElementById
    let para = document.getElementById("demo");
    console.log(para.textContent); // Output: Hello!
  </script>
</body>
```

Common DOM Methods

Method	Description
getElementById("id")	Gets an element by its ID
getElementsByName("class")	Gets elements by class
getElementsByTagName("tag")	Gets elements by tag
querySelector("selector")	Gets the first element matching a CSS selector
createElement("tag")	Creates a new element
appendChild(node)	Adds a node as the last child
removeChild(node)	Removes a child node

Q5(b)

Write a javascript program to show handling of events from textbox and password elements.

```
<!DOCTYPE html>
<html>
<head>
  <title>Textbox and Password Event Handling</title>
  <style>
    body {
      font-family: Arial;
      padding: 20px;
    }
    input {
      display: block;
      margin-bottom: 10px;
      padding: 8px;
      width: 250px;
    }
    #output {
      margin-top: 15px;
      font-weight: bold;
      color: green;
    }
  </style>
</head>
<body>

  <h2>Event Handling Demo</h2>
```

10

L3

CO3

```

<label for="username">Username:</label>
<input type="text" id="username" placeholder="Enter username">

<label for="password">Password:</label>
<input type="password" id="password" placeholder="Enter password">

<div id="output"></div>

<script>
const usernameInput = document.getElementById('username');
const passwordInput = document.getElementById('password');
const outputDiv = document.getElementById

```

Q6(a) Briefly describe Window objects properties and methods.

Brief Description of window Object Properties and Methods in JavaScript

The **window object** is the **global object** in a web browser that represents the current browser window or tab. It provides access to browser features such as dialogs, timers, screen information, URL, storage, and more.

♦ Common Properties of the window Object

Property	Description
window.document	Refers to the DOM (HTML document).
window.location	Contains info about the current URL.
window.innerWidth	Width of the browser viewport in pixels.
window.innerHeight	Height of the browser viewport in pixels.
window.navigator	Information about the user's browser.
window.screen	Screen-related info (height, width, etc.).
window.history	Allows navigation through browser history.
window.localStorage	Stores data with no expiration (persistent).
window.sessionStorage	Stores data until the browser tab is closed.

♦ Common Methods of the window Object

Method	Description
--------	-------------

10 L2 CO3

alert(message)	Displays an alert dialog box.			
confirm(message)	Shows OK/Cancel dialog and returns true or false.			
prompt(message)	Displays a prompt asking for user input.			
open(url)	Opens a new browser window or tab.			
close()	Closes the current window (if opened by script).			
setTimeout(func, ms)	Executes a function after a delay.			
setInterval(func, ms)	Repeats a function at regular intervals.			
print()	Opens the browser print dialog.			

Discuss Event handling. Explain it with an example.			
<p>BASIC CONCEPTS OF EVENT HANDLING</p> <ul style="list-style-type: none">One important use of JavaScript for Web programming is to detect certain activities of the browser and the browser user and provide computation when those activities occur. These computations are specified with a special form of programming called event- driven programming.In conventional (non-event-driven) programming, the code itself specifies the order in which it is executed, although the order is usually affected by the program’s input data.In event-driven programming, parts of the program are executed at completely unpredictable times, often triggered by user interactions with the program that is executing.An event is a notification that something specific has occurred, either with the browser, such as the completion of the loading of a document, or because of a browser user action, such as a mouse click on a form button.An event handler is a script that is implicitly executed in response to the appearance of an event. Event handlers enable a Web document to be responsive to browser and user activities.One of the most common uses of event handlers is to check for simple errors and omissions in user input to the elements of a form, either when they are changed or when the form is submitted.This kind of checking saves the time of sending incorrect form data to the server.Because events are JavaScript objects, their names are case sensitive. The names of all event objects have only lowercase	10	L2	CO3

letters.

- Events are created by activities associated with specific XHTML elements.
- The process of connecting an event handler to an event is called registration.
- There are two distinct approaches to event handler registration, one that assigns tag attributes and one that assigns handler addresses to object properties.

EVENTS, ATTRIBUTES, AND TAGS

Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

In many cases, the same attribute can appear in several different tags.

The circumstances under which an event is created are related to a tag and an attribute, and they can be different for the same attribute when it appears in different tags.

Q7(a)

Briefly explain the following with examples:

1. AngularJS Number
2. AngularJS Strings
3. AngularJS Objects
4. AngularJS Arrays

1. AngularJS Numbers

AngularJS handles numbers like regular JavaScript. You can use them for calculations, display, or data binding.

Example:

html

10

L2

CO4

```
CopyEdit
<div ng-app="" ng-init="num1=10; num2=5">
  Sum: {{ num1 + num2 }}
</div>
```

Output:-

This will display: Sum: 15

2. AngularJS Strings

Strings in AngularJS are just like in JavaScript. You can use them for names, messages, etc.

Example:

```
html
CopyEdit
<div ng-app="" ng-init="message='Hello, AngularJS!'">
  {{ message }}
</div>
```

Output:-

This will display: Hello, AngularJS!

3. AngularJS Objects

Objects in AngularJS are collections of key-value pairs. You can bind and display object properties using expressions.

Example:

```
html
CopyEdit
<div ng-app="" ng-init="student={name:'Daya', age:25}">
  Name: {{ student.name }} <br>
  Age: {{ student.age }}
</div>
```

Output:-

This will display the name and age from the student object.

4. AngularJS Arrays

Arrays are ordered lists of data. You can iterate over them using ng-repeat.

Example:

```
html
CopyEdit
<div ng-app="" ng-init="fruits=['Apple', 'Banana', 'Mango']">
  <ul>
    <li ng-repeat="fruit in fruits">{{ fruit }}</li>
  </ul>
</div>
```

Output:-

This will display a list of fruits.

Q7(b)

Discuss the use of filters in AngularJS with an example.

Filters

What is filter?

Filter is used to format the value of data. The pipe sign (|) indicates that filter is used. The proper syntax of filter looks like this:

Value | filter

Let`s try to understand the filters one by one.

Uppercase filter

Value | uppercase

The uppercase filter changes the text to upper case. Suppose a user writes a text in lower case (e.g. ray) or title case (e.g. Ray) or in mixed case (e.g. rAy or RaY or rAY etc.), and you want the upper case result, then you will have to use upper case filter.

Example 3.1

```
<!DOCTYPE html>
<html >
<head>
  <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/> </head>
<body>
<h3>Using Upper Case Filter</h3> <div ng-app="" ng-init="Username=
'ray' "> <p>User Name: <input type="text" ng-model = "Username"></p>
<p style="color:red" ng-bind="Username | uppercase"></p> </div>
</body>
</html>
```

Output:

Using Upper Case Filter

User Name:

RAY

Explanation:

“Username | uppercase” changes the value of “Username” to uppercase.

10

L2

CO4

In the above example, I set the default value (**ray**) in lower case, but the result becomes upper case (RAY).

Lowercase filter

Value | lowercase

The lowercase filter changes the text to lower case. Suppose a user writes a text in upper case (e.g. RAY YAO) or title case (e.g. Ray Yao) or in mixed case (e.g. rAy or RaY or rAY etc.), and you want the lower case result, then you will have to use lower case filter.

Example 3.2

```
<html >
<head>
  <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> </head>
<body>
<h3>Using Lower Case Filter</h3> <div ng-app="" ng-init="Username=
'Ray YAO' "> <p>User Name: <input type="text" ng-model="Username">
</p> <p style="color:red" ng-bind="Username | lowercase"></p> </div>
</body>
</html>
```

Open the notepad and paste the above mentioned code with .html extension.

Output:

Using Lower Case Filter

User Name:

ray yao

Explanation:

“Username | lowercase”: changes the value of “Username” to lowercase.

In the above example, when I enter text (**Ray YAO**) in upper case, but the result become lower case (ray yao).

OrderBy filter

OrderBy filter is used to display values in ascending order or descending order. The syntax of “orderBy” looks like this:

```
Value | orderBy: 'value' //for ascending order  
Value | orderBy: '-value' //for descending order
```

Let`s take an example for better understanding.

Example 3.3

```
<!DOCTYPE html>  
<html >  
<head>  
  <title>AngularJSfor beginners</title> <script src="js\angular.min.js">  
</script> </head>  
<body>  
<h1>Using OrderBy filter</h1> <div ng-app="" ng-init="StudentsResult=  
  [{name: 'Tienq', marks:81},          {name: 'Svbrf', marks:70},  
  {name: 'Yaito', marks:90},          {name: 'Pewfn', marks:63}, {name:  
  'Riet', marks:98}]"> <table border="1" > <tr>  
  <th>Student Name</th> <th>Mathematics' Result</th> </tr>  
  <tr ng-repeat="x in StudentsResult | orderBy:'-marks' "> <td ng-  
  bind="x.name "></td> <td ng-bind="x.marks "></td> </tr>  
</table>  
</div>  
</body>  
</html>
```

Output:

Using OrderBy filter

Student Name	Mathematics' Result
Riet	98
Yaito	90
Tienq	81
Svbrf	70
Pewfn	63

Explanation:

StudentsResult | orderBy:'-marks' displays the values of StudentsResult in descending order.

You can see that the highest mark is on top and the lowest mark is on bottom by using (value | orderBy:'-marks').

If you want reverser the order, you can remove the “-“sign”.

Currency filter

Value | currency

The currency filter is used to display the result in currency format.

Example 3.5

```
<!DOCTYPE html>
<html >
<head>
  <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/> </head>
<body>
<h1>Using Currency filter</h1> <div ng-app="" ng-init =
"Employees_Monthly_Salary=[{name: 'Jay', salary:8100}, {name: 'Sdwt',
salary:7000}, {name: 'Hao', salary:9000}, {name: 'Luoe',
salary:6300}, {name: 'Fin', salary:9800}]"> <table border="1" > <tr>
<th>Employee Name</th> <th>Employee Salary</th> </tr>
<tr ng-repeat="x in Employees_Monthly_Salary "> <td ng-bind="x.name"
"></td> <td ng-bind="x.salary | currency "></td> </tr>
</table>
</div>
</body>
</html>
```

Open the notepad and paste the above mentioned code with .html extension.

Output:

Using Currency filter

Employee Name	Employee Salary
Jay	\$8,100.00
Sdwt	\$7,000.00
Hao	\$9,000.00
Luoe	\$6,300.00
Fin	\$9,800.00

Explanation:

"x.salary | currency " converts the salary to currency format.

In the above example, there are two columns in the table, the first column is Employee Name and the second is Employee Salary. The salary column displays the salary in currency format.

Array filter

Array | filter:input

“Array | filter:input” can filter the array elements based on the user input.

Example 3.6

```
<!DOCTYPE html>
<html ng-app="">
<head>
<script src="js/angular.min.js"></script> <meta charset="utf-8"> </head>
<body>
<div ng-init="students =    // define an array “students”
[{name:'Andy', age:'19'},
 {name:'Rose', age:'18'},
 {name:'Jony', age:'17'},
 {name:'Judy', age:'16'},
 {name:'Tomy', age:'15'},
 {name:'Lily', age:'14'}]"> </div>

<table>
  <tr><th>Name</th><th>Age</th></tr> <tr ng-repeat="person in students |
filter:myList" > // filter the array “students” according to the input value
  <td>{{ person.name }}</td> <td>{{ person.age }}</td> </tr>
</table>
<br><br>
<label>Please input one of the above name or age <br><br>
<input ng-model="myList"> </label>  // user input </body>
</html>
```

Please try to input a number 18 to text field.

Output:

Name Age
Rose 18

Please input one of the above name or age

Explanation:

“<div ng-init=“students =...” defines an array “students”.

“person in students | filter:myList” filters the array “students” according to the input value “myList”.

<input ng-model=“myList”> accepts the user input, and store the input value to “myList”.

When you input 18 to text field, the output shows “Rose 18”.

Q8(a)	<p>What is Angular JS? Explain the following AngularJS directives: (i) ng_app (ii) ng_model (iii) ng_bind</p> <p>❖ What is AngularJS?</p> <p>AngularJS is a JavaScript-based front-end framework developed by Google. It is used to create dynamic, single-page web applications (SPAs). AngularJS extends HTML by adding new attributes called directives, and binds data to HTML using expressions.</p> <p>♦ Key Features of AngularJS:</p> <ul style="list-style-type: none"> • Two-way data binding • MVC architecture • Dependency injection • Directives for dynamic behavior • Templating using HTML <p>❖ Explanation of AngularJS Directives:</p> <p>(i) ng-app</p> <p>Purpose: Defines the root element of an AngularJS application. It tells AngularJS where to start compiling and initializing the app.</p> <p>Example:</p> <pre>html CopyEdit <div ng-app=""> <p>My First AngularJS App</p> </div></pre> <p>Output:-</p> <p><i>AngularJS will activate within this <div>.</i></p> <p>(ii) ng-model</p> <p>Purpose: Binds the value of an HTML control (input, select, textarea) to a variable in the AngularJS application.</p> <p>Example:</p> <pre>html CopyEdit <div ng-app="" ng-init="name='Daya'"> <input type="text" ng-model="name"> <p>You entered: {{ name }}</p> </div></pre>	10	L2	CO4
-------	--	----	----	-----

Output:-

As you type in the input box, the value updates in real time.

(iii) ng-bind

Purpose:

Binds the value of an expression to the inner text of an HTML element (like {{ expression }} , but cleaner and safer).

Example:

```
html
CopyEdit
<div ng-app="" ng-init="course='MCA'">
  <p ng-bind="course"></p>
</div>
```

Output:-

This will display: MCA

Q8(b)

Explain AngularJS expressions. Write an Angular JS program to use expressions.

Expressions

{{ Expression }}

{{Expression}} is used to bind the value with html element and displays the value. It works same as **ng-bind** directive. {{Expression}} is written within two curly brackets. The {{expression}} is basically pure JavaScript expression.

10

L3

CO4

String Expression

We know that string is collection of characters. In AngularJS the string expression looks like this.

```
<element> {{First String + Second String}}
</element>
```

Example 6.1

```
<!DOCTYPE html>

<html >

<head>

    <title>AngularJS for beginners</title> <script src="js/angular.min.js">
</script> </head>

<body>

<h4>Combine Two String Using String Expression</h4> <div ng-app="" >
First String &nbsp;&nbsp;&nbsp;; <input type="text" ng-
model="firstString"/><br><br> Second String: <input ng-
model="secondString"/><br><br> Resulting String:<p
style="color:blue;font-weight:bold;">{{firstString + " "+secondString}}
</p> </div>

</body>

</html>
```

Output:

Combine Two String Using String Expression

First String : Ray

Second String: Yao

Resulting String:

Ray Yao

Explanation:

“{firstString + " " + secondString}” joins two strings together.

`{{ expression }}` displays the value of expression.

In the above example, the text **Ray** is written in the first text box and **Yao** in the second text box, but in the resulting string area, Ray string and Yao string are combined due to use of string expression `{{ }}`. Note: The plus `+` sign is used for string concatenation.

Number Expression

In AngularJS you can perform different mathematic operation by using Number Expression.

$$\langle \text{element} \rangle \{ \{ \text{First Number} + \text{Second Number} \} \}$$

Example 6.2

```
<!DOCTYPE html>
<html >
<head>
    <title>AngularJS for beginners</title> <script src="js/angular.min.js">
</script> </head>
<body>
<h4>Multiply Two Number Using Number Expression</h4> <div ng-app=""
ng-init="firstNumber=9;secondNumber=6"> First Number
&nbsp;&nbsp;&nbsp;; <input type="number" ng-model="firstNumber"/>
<br><br> Second Number: <input type="number" ng-
model="secondNumber"/><br><br> Result:<p style="color:blue;font-
weight:bold;">{{firstNumber * secondNumber}}</p> </div>
</body>
</html>
```

Output:

Multiply Two Number Using Number Expression

First Number : 9

Second Number: 6

Result:

54

Explanation:

“{{firstNumber * secondNumber}}” multiplies the firstNumber and the secondNumber.

`{{ expression }}` displays the value of expression.

In the above example, the number **9** is written in the first text box and **6** in the second text box, and **54** is the result of multiplication of 9 and 6. You can perform any arithmetic operation by using Number Expression.

Object Expression

AngularJS object works like a JavaScript object. The syntax looks like this:

```
object = {property: value}
```

Example 6.3

```
<html >
<script src= "js\angular.min.js"></script>
<body>
<h4>Object Expression</h4> <div ng-app="" ng-init="EmployeeObject =
{Emp_name: 'Jay Smith',  Emp_Month: 'June.15 2015', Emp_salary:
'$8000'}"> <p>Employee Name : {{EmployeeObject.Emp_name}}</p>
<p>Salary's Month: {{EmployeeObject.Emp_Month}}</p> <p>Employee
Salary: {{EmployeeObject.Emp_salary}}</p> </div>
</body>
</html>
```

Output:

Object Expression

```
Employee Name: Jay Smith
Salary's Month: June 15 2015
Employee Salary: $8000
```

“Emp_salary” is a property.

{{ object.property }} displays the value of the property.

Array Expression

The array expression of AngularJS works like JavaScript array. The syntax looks like this:

```
Array=[val1, val2, val3,]
```

Example 6.4

```
<!DOCTYPE html>
```

```
<html >
```

```
<head>
```

```
  <title>AngularJS for beginners</title> <script src="js/angular.min.js">
```

```
</script> </head>
```

```
<body>
```

```
<h4>My Math Result Using Array Expression</h4> <div ng-app="" ng-  
init="MyArray=[98,96,93,90,99]"> <p>My score in mathematics is:  
{MyArray[4]}</p> </div>
```

```
</body>
```

```
</html>
```

Output:

My Math Result Using Array Expression

My score in mathematics is: 99

Explanation:

“MyArray=[98,96,93,90,99]” is an array.

“{{MyArray[4]}}” displays the value whose index is 4 in MyArray.

Q9(a) What is AngularJS Services? Explain Them with examples.

Services

Angular Service Service is a function or an object, which is used to provide with a specified action. In AngularJS, there are about 30 builtin services, such as \$http, \$location, \$interval and \$timeout.

Types of Services in AngularJS

AngularJS provides several built-in services and also allows you to create custom services. Here are some commonly used built-in services:

1. **\$http**: For making AJAX requests.
2. **\$location**: For handling URL manipulation.
3. **\$timeout**: For delaying code execution.
4. **\$interval**: For repeated execution at specified intervals.

10

L2

CO4

Creating a Custom Service

Custom services can be created using:

1. **Factory**
2. **Service**
3. **Provider**

1. Using Factory

A factory function returns an object or a function that is injected where needed.

Example:

```
app.factory('mathService', function() {  
  return {  
    add: function(a, b) {  
      return a + b;  
    },  
    subtract: function(a, b) {  
      return a - b;  
    }  
  };  
});
```

//Usage in a Controller:

```
// Inject the Service into a Controller  
app.controller('mathController', function($scope, mathService) {  
  $scope.addition = mathService.add(10, 5); // 15  
  $scope.subtraction = mathService.subtract(10, 5); // 5  
});
```

Explanation:-

1. **Factory (mathService)** defines two methods:
 - add(a, b) → Returns the sum.
 - subtract(a, b) → Returns the difference.
2. **Controller (mathController)** calls these functions with 10 and 5, storing the results in \$scope.addition and \$scope.subtraction.
3. The results are displayed in `<p>{{ addition }}</p>` and `<p>{{ subtraction }}</p>`.

2. Using Service

In AngularJS, a **service** is used to **share reusable code** across different components like controllers, directives, or even other services

Example:

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

```
// Simple Service that returns a static message
```

```
app.service('messageService', function() {  
    this.getMessage = function() {  
        return "Hello from AngularJS Service!";  
    };  
});
```

```
// Controller using the Service
```

```
app.controller('messageController', function($scope, messageService) {  
    $scope.message = messageService.getMessage();  
});
```

Explanation:-

1. **Service (messageService)** defines a function `getMessage()` that returns "Hello from AngularJS Service!".
2. **Controller (messageController)** calls `messageService.getMessage()` and assigns the result to `$scope.message`.
3. The message **"Hello from AngularJS Service!"** is displayed inside `<p>{{ message }}</p>`.

3. Using Provider

A provider gives you the most control over service creation. It is used when you need to configure a service before making it available.

Example:

```
app.provider('messageService', function() {  
    var prefix = "";  
  
    this.setPrefix = function(value) {  
        prefix = value;  
    };  
  
    this.$get = function() {
```



```

        return {
            getMessage: function(message) {
                return prefix + " " + message;
            }
        };
    };
});

// Configure the provider
app.config(function(messageServiceProvider) {
    messageServiceProvider.setPrefix('Hello');
});

app.controller('messageController', function($scope, messageService) {
    $scope.message = messageService.getMessage('AngularJS');
});

```

Explanation:-

1. **Provider (messageService)**
 - Has a variable prefix that stores a default greeting ("Hello").
 - Has a method setPrefix(value) to change the prefix.
 - Implements \$get() that returns the actual service.
2. **Configuration Phase (app.config)**
 - Changes the prefix to "Welcome".
3. **Controller (messageController)**
 - Calls messageService.getMessage('AngularJS') to generate "Welcome, AngularJS!".
4. **Output on the page:**
 - **"Welcome, AngularJS!"** is displayed.

Built-in AngularJS Services

1. \$http (For AJAX requests)

Used to communicate with a server.

Example:

```

<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></
script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>Today's welcome message is:</p>

<h1>{{myWelcome}}</h1>

</div>

<p>The $http service requests a page on the server, and the response is set as
the value of the "myWelcome" variable.</p>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
    $http.get("welcome.htm").then(function (response) {
        $scope.myWelcome = response.data;
    });
});
</script>

</body>
</html>

```

2. \$timeout (For Delayed Execution)

Executes a function after a delay.

Example:

```

<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></
script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">

```

<p>This header will change after two seconds:</p>

<h1>{{myHeader}}</h1>

</div>

<p>The \$timeout service runs a function after a specified number of milliseconds.</p>

<script>

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $timeout) {
  $scope.myHeader = "Hello World!";
  $timeout(function () {
    $scope.myHeader = "How are you today?";
  }, 2000);
});
```

</body>

</html>

3. \$interval (For Repeated Execution)

Executes a function repeatedly at a specified time interval.

Example:

| | | | | |
|-------|--|----|----|-----|
| | <pre> <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></ script> <body> <div ng-app="myApp" ng-controller="myCtrl"> <p>The url of this page is:</p> <h3>{{myUrl}}</h3> </div> <p>This example uses the built-in \$location service to get the absolute url of the page.</p> <script> var app = angular.module('myApp', []); app.controller('myCtrl', function(\$scope, \$location) { \$scope.myUrl = \$location.absUrl(); }); </script> </body> </html> </pre> | | | |
| Q9(b) | <p>Write an Angular JS program to demonstrate client-side form validation.</p> <p>AngularJS Program to Demonstrate Client-Side Form Validation</p> <p>In this example, we'll create a simple AngularJS form with client-side validation. We'll validate that the user enters a valid name, email, and password before submitting the form.</p> <ul style="list-style-type: none"> ❖ Key Features of AngularJS Form Validation: <ul style="list-style-type: none"> ● ng-required: Makes a field required. ● ng-pattern: Validates the field based on a regular expression. ● ng-minlength / ng-maxlength: Sets a minimum and maximum length for text input. ● ng-model: Binds input fields to model properties. | 10 | L3 | CO4 |

- **\$valid / \$invalid**: Indicates whether the form is valid or not.

❖ **AngularJS Client-Side Form Validation Example:**

```
<!DOCTYPE html>
<html>
<head>
  <title>AngularJS Form Validation</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></scri
pt>
</head>
<body>

<div ng-app="formApp" ng-controller="formCtrl">

  <h2>AngularJS Client-Side Form Validation Example</h2>

  <!-- Form -->
  <form name="userForm" ng-submit="submitForm(userForm)" novalidate>

    <!-- Name -->
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" ng-model="user.name"
ng-required="true" />
    <span style="color: red" ng-show="userForm.name.$touched &&
userForm.name.$invalid">Name is required.</span>
    <br><br>

    <!-- Email -->
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" ng-model="user.email"
ng-required="true" ng-pattern="/^[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$/">
    <span style="color: red" ng-show="userForm.email.$touched &&
userForm.email.$invalid">Enter a valid email.</span>
    <br><br>

    <!-- Password -->
    <label for="password">Password:</label>
    <input type="password" id="password" name="password"
ng-model="user.password" ng-required="true" ng-minlength="6"
ng-maxlength="12" />
    <span style="color: red" ng-show="userForm.password.$touched &&
userForm.password.$invalid">Password must be between 6 and 12
characters.</span>
    <br><br>

    <!-- Submit Button -->
    <button type="submit" ng-disabled="userForm.$invalid">Submit</button>

  </form>

  <div ng-if="formSubmitted">
    <h3>Form Submitted Successfully!</h3>
    <p>Name: {{ user.name }}</p>
    <p>Email: {{ user.email }}</p>
    <p>Password: {{ user.password }}</p>
  </div>
```

	<pre> </div> <script> // AngularJS Application and Controller var app = angular.module('formApp', []); app.controller('formCtrl', function(\$scope) { // Form submission logic \$scope.submitForm = function(form) { if (form.\$valid) { \$scope.formSubmitted = true; } else { alert('Please fill out the form correctly.');</pre>			
Q10(a)	<p>Briefly explain about AngularJS Events with an example.</p> <p><u>Events</u></p> <p>Event</p> <p>Events are associated with different HTML elements. e.g. the click event is associated with button element; similarly keypress event is associated with text box or text area element. AngularJS provides multiple events which are associated with HTML control.</p>	10	L3	CO4

Click event

ng-click = "expression"

ng-click = "expression" defines a click event. When a button is clicked, an event occurs, and evaluates the expression. The click event normally works on button.

Example 5.1

```
<!DOCTYPE html>
<html >
<head>
  <title>AngularJSfor beginners</title> <script src="js\angular.min.js">
</script> </head>
<body>
<h3>Add Two Numbers Using Click Event</h3> <div ng-app="" ng-
init="firstNumber=47; secondNumber=23"> <p>First Number : <input
type="number" ng-model = "firstNumber"></p> <p>Second Number:
<input type="number" ng-model = "secondNumber"></p> <button ng-
click="Result=firstNumber + secondNumber"> Add Numbers </button>
<p>Result:<p style="font-weight:bold; color:blue" ng-bind = "Result">
</p></p> </div>
</body>
</html>
```

Open the notepad and paste the above mentioned code with .html extension.

Output:

Add Two Numbers Using Click Event

First Number :

Second Number:

Result:

70

Explanation:

“<button ng-click="Result=firstNumber + secondNumber"> Add Numbers </button>”: when the button is clicked, an event occurs. “firstNumber” add “secondNumber”, and assigns the result to “Result”.

Mouse Move event

```
ng-mousemove = "expression"
```

ng-mousemove = "expression" defines a mouse move event. When the mouse moves, an event occurs, and evaluates the expression.

Mouse move event normally works on div, body and specific area or element

Example 5.3

```
<!doctype html>
<html>
<head>
<script src="js\angular.min.js"> </script>
</head>
<body ng-app="">
<br><br>
<textarea ng-mousemove="count = count + 1"
ng-init="count=0">
```

Here is a textarea

```
</textarea>
<br><br>
<h2>count: {{count}}</h2> </body>
</html>
```

(Assume you move the mouse on the textarea for 20 times.) **Output:**

```
Here is a textarea.
```

count: 20

Explanation:

"ng-mousemove="count = count + 1" : when mouse moves on the textarea, "count" increases 1.

"ng-init="count=0" initializes the "count" value as 0.

{{count}} displays the value of "count".

"count : 20" means that mouse moves for twenty times.

Mouse Over event

```
ng-mouseover = "expression"
```

ng-mouseover = "expression" defines a mouse over event. When the mouse hovers over, an event occurs, and evaluates the expression.

Mouse over event normally works on div, body and specific area or element

Example 5.4

```
<!doctype html>
<html>
<script src="js\angular.min.js"></script> <body ng-app="">
  <br><br>
  <textarea ng-mouseover="count = count + 1"
  ng-init="count=0">
    Here is a textarea.
  </textarea>
  <br><br>
  <h2>count: {{count}}</h2> </body>
</html>
```

(Assume you move the mouse over the textarea for 2 times.) **Output:**

```
Here is a textarea.
```

count: 2

Explanation:

"ng-mouseover="count = count + 1"" : when mouse moves over the textarea "count" increases 1.

"ng-init="count=0"" initializes the "count" value as 0.

{{count}} displays the value of "count".

"count : 2" means that mouse moves over the textarea for two times.

Mouse Leave event

```
ng-mouseleave = "expression";
```

ng-mouseleave = "expression" defines a mouse leave event. When the mouse leaves a specified element, an event occurs, and evaluates the expression.

Example 5.5

```
<!doctype html>
<html>
<head>
<script src="js/angular.min.js"> </script>
</head>
<body ng-app=""> <br><br>
<textarea ng-mouseleave="count = count + 1"
ng-init="count=0"> Here is a textarea
</textarea>
<br><br> <h2>count: {{count}}</h2> <body>
</html>
```

(Assume you move the mouse and leave the textarea for 10 times.).

Output:

```
Here is a textarea.
```

count: 10

Explanation:

"ng-mouseleave="count = count + 1"" : when mouse leaves the textarea,

"count" increases 1.

"ng-init="count=0"" initializes the "count" value as 0.

{{count}} displays the value of "count".

"count : 10" means that mouse leaves ten times.

Key Up event

```
ng-keyup = "expression";
```

ng-keyup = "expression" defines a key up event. When the key is up in specified element, an event occurs, and evaluates the expression.

Key up event normally works on text box and text area.

Example 5.6

```
<!doctype html>
<html>
<head>
<script src="js/angular.min.js"> </script>
</head>
<body ng-app="">
<br><br>
<textarea ng-keyup="count = count + 1"
ng-init="count=0">
Here is a textarea
</textarea>
<br><br>
<h2>count: {{count}}</h2> <body>
</html>
```

(Assume that you type 12345678 in the textarea.) **Output:**

```
Here is a textarea.
```

```
12345678
```

count: 8

Explanation:

“ng-keyup=“count = count + 1”” : when typing something and key up on the textarea, “count” increases 1.

“ng-init=“count=0”” initializes the “count” value as 0.

{{count}} displays the value of “count”.

“count : 8” means that the typing makes key up 8 times.

Key Down event

```
ng-keydown = "expression";
```

ng-keyup = "expression" defines a key down event. When the key is down in a specified element, an event occurs, and evaluates the expression.

Key down event normally works on text box and text area..

Example 5.7

```
<!doctype html>
<html>
<head>
<script src="js\angular.min.js"> </script>
</head>
<body ng-app="">
<br><br>
<textarea ng-keydown="count = count + 1"
ng-init="count=0">
Here is a textarea
</textarea>
<br><br>
<h2>count: {{count}}</h2> <body>
</html>
```

(Assume that you type 123456 in the textarea.) **Output:**

Here is a textarea.

123456

count: 6

Explanation:

"ng-keydown="count = count + 1"" : when typing something and key down on the textarea, "count" increases 1.

"ng-init="count=0"" initializes the "count" value as 0.

{{count}} displays the value of "count".

"count : 6" means that the typing makes key down 6 times.

Q10
(b)

Explain Angular JS Forms and its elements.

Form:-

AngularJS facilitates you to create a form enriches with data binding and validation of input controls.

Input controls are ways for a user to enter data. A form is a collection of controls for the purpose of grouping related controls together.

10

L3

CO4

Following are the input controls used in AngularJS forms:

- o input elements
- o select elements
- o button elements
- o textarea elements

AngularJS provides multiple events that can be associated with the HTML controls. These events are associated with the different HTML input elements.

Data-Binding

Input controls provides data-binding by using the ng-model directive.

```
<input type="text" ng-model="firstname">
```

The application does now have a property named firstname.

The ng-model directive binds the input controller to the rest of your application.

The property firstname, can be referred to in a controller:

Example

```
<script>
var app=angular.module('myApp',[]);
app.controller('formCtrl', function($scope){
    $scope.firstname = "John";
});
</script>
```

It can also be referred to elsewhere in the application:

Example

```
<form>
FirstName: <input type="text" ng-model="firstname">
</form>
```

```
<h1>You entered: {{firstname}}</h1>
```

Checkbox

A checkbox has the value true or false. Apply the ng-model directive to a checkbox, and use its value in your application.

Example

Show the header if the checkbox is checked:

```
<form>
Check_to_show_a_header:
<input type="checkbox" ng-model="myVar">
```

```
</form>
```

```
<h1 ng-show="myVar">My Header</h1>
```

Radio Buttons

Bind radio buttons to your application with the ng-model directive.

Radio buttons with the same ng-model can have different values, but only the selected one will be used.

Example

Display some text, based on the value of the selected radio button:

```
<form>
```

Pick_a_topic:

```
<input type="radio" ng-model="myVar" value="dogs">Dogs
```

```
<input type="radio" ng-model="myVar" value="tuts">Tutorials
```

```
<input type="radio" ng-model="myVar" value="cars">Cars
```

```
</form>
```

The value of myVar will be either dogs, tuts, or cars.

Selectbox

Bind select boxes to your application with the ng-model directive.

The property defined in the ng-model attribute will have the value of the selected option in the select box.

Example

Display some text, based on the value of the selected option:

```
<form>
```

Select_a_topic:

```
<select ng-model="myVar">
```

```
<option value="">
```

```
<option value="dogs">Dogs
```

```
<option value="tuts">Tutorials
```

```
<option value="cars">Cars
```

```
</select>
```

```
</form>
```

Example:-

```
<!DOCTYPE html>
```

```
<html lang="en">
```



```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.mi
n.js"></script>
<body>

<div ng-app="myApp" ng-controller="formCtrl">
  <form novalidate>
    First Name:<br>
    <input type="text" ng-model="user.firstName"><br>
    Last Name:<br>
    <input type="text" ng-model="user.lastName">
    <br><br>
    <button ng-click="reset()">RESET</button>
  </form>
  <p>form = {{user}}</p>
  <p>master = {{master}}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
  $scope.master = {firstName:"John", lastName:"Doe"};
  $scope.reset = function() {
    $scope.user = angular.copy($scope.master);
  };
  $scope.reset();
});
</script>

</body>
</html>
```

Output:-

FirstName:

LastName:

RESET

form = {"firstName":"John","lastName":"Doe"}

master = {"firstName":"John","lastName":"Doe"}