K. K. Wagh Institute of Engineering Education and Research, Nashik

(An Autonomous Institute from A. Y. 2022-23)

Model Answer Key

Exam Seat No.:

End-Sem Examination-I — Winter 2025

Academic Year: 2025-26 — Sem: I

Class: F.Y — Program: MCA

Branch Code:09 — Pattern: 2022

Name of Course: Web Technology — Course Code: MCA221004

Max. Marks: 60 — Duration: 2:30 Hrs.

Instructions: Candidates should read carefully the instructions printed on the Question Paper and on the cover page of the Answer Book, which is provided for their use.

(*keep space*)

1. This question paper contains 2 page(s).
2. Answer to each new question is to be started on a new page.
3. Assume suitable data wherever required, but justify it.
4. Draw the neat labelled diagrams, wherever necessary.
5. The last columns indicate the Course Outcome.

Q. No.	Details	Max. Marks	CO No.	BT Level
Q.1	Explain how headings, paragraphs, line breaks, colors, and fonts are used in HTML5 (give a one-line example for each). Ans: In HTML5, formatting and text styling are achieved using semantic tags and attributes that describe how content should appear in a webpage. **1. Headings:** HTML provides six heading levels <h1> to <h6> to define titles and sections. **Example:** <h1>Main Title</h1> **2. Paragraphs:** The tag is used to group text into readable paragraphs. **Example:** This is a paragraph. **3. Line Breaks:** The tag inserts a single line break without creating a new paragraph. **Example:** Hello World **4. Colors:** Colors can be applied using the style attribute with color names, RGB, or HEX values. **Example:** Blue Text **5. Fonts:** Fonts are applied using CSS properties like font-family, font-size, or font-style.	[6]	CO1	L2

	Example: Sample Font			

Q.2	Apply JavaScript number, string, and array properties to create a small script that prints one example of each.
	Ans:
	
	JavaScript provides built-in properties for its data types to help programmers perform operations easily.
	
	A **number property** such as Number.MAX_VALUE returns the largest possible number JavaScript can store. This is useful in calculations where limits are required.
	
	A **string property** like length returns the number of characters in a string. It is commonly used in validations and parsing.
	
	An **array property** like length helps in determining how many elements are stored inside an array, which is useful for loops.
	
	Example Script
	
	console.log("Max Number = ", Number.MAX_VALUE);
	console.log("String Length = ", "JavaScript".length);
	console.log("Array Length = ", [10,20,30,40].length);
	
	These properties demonstrate efficient data handling in JavaScript.
		[6]	CO2	L3

Q.3	a) Apply XML rules to create a small XML document showing structure, entity references, and a DTD. (8 Marks) Ans: XML is a markup language designed to store and transport data. It is strict because it must follow rules like: 1. One root element 2. Case-sensitive tags 3. Tags must be properly nested 4. All elements must be closed Entity references (e.g., &, <) are used to display special characters. A DTD (Document Type Definition) validates structure by defining allowed elements. **XML Document** <?xml version="1.0"?> <!DOCTYPE student SYSTEM "student.dtd">	[16]	CO3	L3

	<student> <name>John & Peter</name> <course>BCA</course> </student> **DTD File (student.dtd)** <!ELEMENT student (name, course)> <!ELEMENT name (#PCDATA)> <!ELEMENT course (#PCDATA)> This XML file is fully well-formed and valid. **OR** b) Use the differences between XML and HTML to reorganize a sample webpage into XML format. (8 Marks) Ans: HTML is designed for displaying data while XML is designed for storing and transporting data. Differences include: • HTML has predefined tags; XML uses user-defined tags. • HTML focuses on appearance; XML focuses on structure. • HTML is not strict; XML must be well-formed. • HTML ignores white space; XML preserves it. **Sample HTML** Student Info Name: John **Converted XML** <student> <details> <name>John</name> </details> </student> This version becomes structured and machine-readable.			
	c) Apply JSON concepts to create a JSON file and show how element nesting and acceptable values work. (8 Marks) Ans: JSON (JavaScript Object Notation) is a lightweight, human-readable data format used for data exchange. It uses key-value pairs and supports various data types:		CO3	L3

- string
- number
- boolean
- array
- object
- null

Nesting means placing objects inside other objects, commonly used in web APIs.

Sample JSON

```
{
  "student": {
    "name": "Aarav",
    "age": 21,
    "active": true,
    "subjects": ["Math", "DBMS", "Java"],
    "marks": {
      "internal": 45,
      "external": 80
    }
  }
}
```

This demonstrates nesting and multiple data types in a single JSON structure.

OR

d) Use JSON formatting methods to document a JSON element, showing an alternative to standard indenting. (8 Marks)
Ans:

JSON formatting refers to how JSON data is displayed so it becomes easier to read, understand, and document. Normally JSON uses standard indentation, usually 2 or 4 spaces, to show nested levels. However, JSON can also be presented using alternative formatting methods such as minified format, custom spacing, or single-line formatting for documentation.

JavaScript provides the method **JSON.stringify()**, which includes formatting arguments.
JSON.stringify(value, replacer, space)

- The **space** parameter defines how many spaces to use for indentation.
- It can be replaced with custom characters to create alternative formats.

	Example with Custom Formatting (Alternative to Standard Indent)
	
	let student = { name: "Riya", age: 20, course: "BCA" };
	
	let formatted = JSON.stringify(student, null, "--");
	console.log(formatted);
	
	Output:
	
	{
	--"name": "Riya",
	--"age": 20,
	--"course": "BCA"
	}
	
	Here, "--" replaces normal spaces, showing an **alternate documentation style**.
	
	Minified JSON (Another Alternative Format)
	
	{"name":"Riya","age":20,"course":"BCA"}
	
	This format removes all spaces, useful for compact storage and transmission.
	
	Thus, JSON formatting methods help document JSON in various styles beyond the standard indentation format.			

Q.4	a) Apply PHP basics to build a small web page showing strings, numbers, and control statements. (8 marks)
	Ans:
	
	PHP is a server-side scripting language used to create dynamic webpages. It supports variables, loops, decision structures, and HTML integration.
	
	String and number variables store text and numeric values. Control statements like if...else are used for decision making.
	
	Example PHP Program
	
	<?php
	$name = "Rahul"; // string
	$age = 20; // number
	
	echo "Name: $name
	";
	echo "Age: $age
	";
	
	if($age >= 18){
	  echo "Status: Eligible to Vote";
		[16]	CO4	L3

```
} else {
   echo "Status: Not Eligible";
}
?>
```

This program demonstrates the fundamentals of PHP.

OR

b) Use PHP arrays and functions to process and display a set of user inputs. (8 marks)

Ans:

In PHP, an **array** is a data structure used to store multiple values in a single variable. Arrays are useful for storing user inputs such as marks, names, or form data. A **function** in PHP is a reusable block of code that performs a specific task. Combining arrays and functions allows efficient processing of multiple inputs.

Example: Process User Inputs Using Array + Function

```
<?php
// User inputs (simulating form data)
$marks = [85, 90, 78, 88];

// Function to calculate average
function calculateAverage($data) {
   return array_sum($data) / count($data);
}

// Display all inputs
echo "Marks Entered: ";
foreach($marks as $m){
   echo $m . " ";
}

// Display processed output
echo "<br>Average Marks: " . calculateAverage($marks);
?>
```

★ **Explanation**

- The array $marks stores multiple user inputs.
- The function calculateAverage() processes the array by calculating the average.
- foreach loop displays each input to the user.
- Finally, the processed result (average marks) is printed.

This demonstrates how PHP arrays and functions work

	together to efficiently handle and display user input.			
	c) Apply web-app architecture rules to arrange the structure of a simple PHP application. (8marks) Ans: A web-app architecture defines how different parts of a web application are organized so the system becomes easy to maintain, scalable, and reusable. PHP applications commonly follow structured architecture instead of putting all code into one file because separation of concerns improves security and reduces errors. One widely used structure is the **MVC (Model–View–Controller)** pattern. • **Model** handles the data and database operations • **View** represents the user interface (HTML/PHP templates) • **Controller** manages user input and connects the Model and View This architecture helps divide responsibilities and reduces code complexity. **Typical PHP Web-App Folder Structure** /project /public index.php /app /controllers UserController.php /models User.php /views login.php dashboard.php /config database.php /assets /css /js /images **Explanation of Parts** • **public/** – Contains index.php, the main entry point accessed by the browser. • **app/controllers/** – Contains PHP files that handle user		CO4	L3

requests, call models, and load views.

- **app/models/** – Contains database logic such as inserting, fetching, or updating data.
- **app/views/** – Contains HTML templates and layout files shown to users.
- **config/** – Stores database connection files and configuration settings.
- **assets/** – Contains CSS, JavaScript, and images used in the interface.

★ **Simple Interaction Example**

index.php

```
require "app/controllers/UserController.php";
$controller = new UserController();
$controller->showLogin();
```

UserController.php

```
class UserController {
   public function showLogin() {
      include "app/views/login.php";
   }
}
```

login.php

```
<h2>Login Page</h2>
<form method="post">
   <input type="text" name="user">
</form>
```

OR

d) Use PHP and SQL together to design a small database and run a basic MySQL query. (8 marks)

Ans:

PHP can interact with MySQL to build data-driven applications. MySQL stores data in tables while PHP sends queries using functions like mysqli_query().

SQL Table Creation

```
CREATE TABLE students(
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(50),
   marks INT
);
```

PHP Code to Run a Query

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

$sql = "SELECT * FROM students";
$result = mysqli_query($conn, $sql);

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

This shows a full PHP-MySQL interaction.

Q.5	a) Apply Angular components, templates, and data binding to build a small interactive UI. (8 marks) Ans: Angular applications are built using components, which contain logic and HTML templates. Templates define the structure of the user interface. Data binding connects the component data with the HTML view. Two-way binding ([(ngModel)]) allows updating data both in the component and the template. **Example** **Component** `@Component({ selector: 'app-demo', template: ` {{message}} <input [(ngModel)]="message"> ` }) export class DemoComponent { message = "Hello Angular!"; }` This creates a simple interactive UI where typing updates the heading in real time. **OR** b) Use NgModule and directives (structure & template) to organize and control an Angular page. (8 marks) Ans:	[16]	CO5	L3

In Angular, an **NgModule** is a container that groups components, directives, services, and pipes together. It helps organize an application into functional units and tells Angular how to load and run the features. The root module is usually AppModule, which imports other modules and declares components used in the application.

Angular also uses **directives** to control how elements behave on a page:

1. Structural Directives

Structural directives change the **layout** of the DOM by adding or removing elements.
Examples:

- *ngIf – displays an element conditionally
- *ngFor – repeats an element for each item
- *ngSwitch – multiple conditional view display

2. Template Directives

Template directives modify the **appearance or behavior** of elements without changing structure.
Examples:

- ngStyle – apply dynamic styles
- ngClass – apply dynamic CSS classes

★ Example: NgModule + Directives

```
// app.module.ts
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  bootstrap: [AppComponent]
})
export class AppModule {}
<!-- app.component.html -->
<div *ngIf="isLoggedIn">Welcome User</div>

<ul>
  <li *ngFor="let item of items" [ngClass]="{'highlight':
item.active}">
    {{ item.name }}
  </li>
</ul>
```

Explanation

- NgModule organizes the component and makes it available to the Angular app.
- *ngIf controls visibility.
- *ngFor structures repeated content.
- ngClass controls styling dynamically.

This shows how NgModule and Angular directives work together to organize and control the UI effectively.

c) Apply Angular forms and services by creating a form and injecting a service to process the data. (8marks)
Ans:

Angular provides FormsModule to collect and validate user input, and **Services** to handle reusable logic such as calculations or data sharing. Using both together allows clean separation of UI and business logic.

Example: Simple Angular Form With Injected Service

service.ts

```
@Injectable({providedIn: 'root'})
export class DataService {
  process(name: string) {
    return "Hello " + name;
  }
}
```

app.component.ts

```
export class AppComponent {
  username = "";
  message = "";

  constructor(private dataService: DataService) {}

  submitForm() {
    this.message = this.dataService.process(this.username);
  }
}
```

app.component.html

```
<input [(ngModel)]="username" placeholder="Enter Name">
<button (click)="submitForm()">Submit</button>

<p>{{ message }}</p>
```

CO5 L3

Explanation

- The form takes user input using **ngModel**.
- The service contains the processing function.
- The component **injects** the service using dependency injection.
- On submit, the service processes data and returns the result.

OR

d) Use API methods (GET, POST, PUT, DELETE) to connect an Angular app to a backend server. (8 marks)
Ans:

Angular connects to backend servers using the HttpClient module, which allows sending HTTP requests. The main CRUD operations are performed using the four common API methods:

- **GET** – fetch data
- **POST** – add new data
- **PUT** – update existing data
- **DELETE** – remove data

Example: Angular Service Calling API Methods

```
@Injectable({ providedIn: 'root' })
export class ApiService {

  apiUrl = "https://example.com/users";

  constructor(private http: HttpClient) {}

  getUsers() {
    return this.http.get(this.apiUrl);          // GET
  }

  addUser(data:any) {
    return this.http.post(this.apiUrl, data);    // POST
  }

  updateUser(id:number, data:any) {
    return this.http.put(`${this.apiUrl}/${id}`, data);   // PUT
  }

  deleteUser(id:number) {
    return this.http.delete(`${this.apiUrl}/${id}`);       //
DELETE
  }
```

	} **Explanation** This service handles all communication with the backend. Components call these functions to interact with the database. Angular's HttpClient makes API interaction simple and follows REST principles.			