BCU BCA 5TH SEM — WEB PROGRAMMING (DCCA511) SOLVED QUESTION BANK

Solved Question Bank by Nikhil P

PART I: SECTION A — 2 MARKS QUESTIONS (PAPERS 1 TO 5)

MODEL QUESTION PAPER – 1

Q1. What is the Internet?

Internet: A global network of interconnected computer systems and autonomous networks that communicate using standardized communication protocols, primarily the TCP/IP protocol suite.
Function: It facilitates worldwide resource sharing, information dissemination, electronic mail, and World Wide Web access across heterogeneous hardware platforms.
Core Protocol: `TCP/IP` (Transmission Control Protocol / Internet Protocol).

Q2. What is a web browser? Give any two examples.

Web Browser: A client application program that retrieves, renders, and displays web resources from web servers by interpreting HTML, CSS, and client-side JavaScript.
Examples: Google Chrome, Mozilla Firefox (also Apple Safari, Microsoft Edge).

Q3. What is HTML? Expand HTML.

Expansion: HTML stands for HyperText Markup Language.
Definition: The standard markup authoring language used to create and structure elements on web pages using predefined tags and attributes.
Syntax: `<p>Welcome to Web Programming</p>`

Q4. What is CSS? Expand CSS.

Expansion: CSS stands for Cascading Style Sheets.
Definition: A stylesheet language used to design, format, and control the visual layout and presentation of HTML documents, separating styling from content.
Syntax: `h1 { color: navy; font-size: 24px; }`

Q5. State any four important features of CSS.

Important Features of CSS:

- **Separation of Structure and Style:** Keeps HTML files clean by isolating formatting rules into separate stylesheets.
- **Reusability and Consistency:** Allows one external stylesheet to apply a uniform layout across multiple web pages.
- **Responsive Design:** Uses media queries to adapt web page layouts dynamically to various screen resolutions.
- **Cascading Rules and Specificity:** Resolves conflicting styling definitions hierarchically based on inline, internal, and external precedence.

Q6. Name the three types of CSS.

Three Types of CSS:

- **Inline CSS:** Applied directly inside an individual HTML tag using the `style` attribute. Example: `<h2 style="color: red;">Text</h2>`
- **Internal (Embedded) CSS:** Declared inside the `<style>` tag within the `<head>` section of an HTML document.
- **External CSS:** Defined in a separate `.css` file and linked using the `<link>` tag. Example: `<link rel="stylesheet" href="style.css">`

Q7. What is JavaScript?

JavaScript: A lightweight, interpreted, object-based client-side scripting language used to create dynamic, interactive web pages.
Role: It manipulates the Document Object Model (DOM), handles browser events (clicks, keypresses), validates user input in forms, and communicates asynchronously with servers via AJAX.
Syntax: `document.getElementById("msg").innerHTML = "Hello World";`

Q8. What is a literal in JavaScript? Name the different types of literals.

Literal: A fixed data value directly specified in the program source code that represents a constant value rather than a variable.
Types of Literals:

- **Numeric Literals:** Integers and floating-point values (e.g., `100`, `3.14`).
- **String Literals:** Characters enclosed in quotes (e.g., `"Bengaluru"`, `'Web'`).
- **Boolean Literals:** `true` and `false`.
- **Object Literals:** Key-value collections (e.g., `{ name: "Nikhil", sem: 5 }`).
- **Array Literals:** Ordered lists of elements (e.g., `["HTML", "CSS", "JS"]`).

Q9. What is server-side scripting?

Server-Side Scripting: A web programming technique where scripts execute on the web server before the generated output is delivered as static HTML to the client browser.
Purpose: It processes database transactions, handles user authentication, maintains session state, and secures core business logic by keeping source code hidden on the server.
Examples: PHP, Node.js, Python, ASP.NET.

Q10. What is PHP? Expand PHP.

Expansion: PHP stands for PHP: Hypertext Preprocessor (a recursive acronym).
Definition: An open-source, interpreted server-side scripting language embedded within HTML to develop dynamic, database-driven web applications.
Syntax:

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

MODEL QUESTION PAPER – 2

Q1. What is the World Wide Web (WWW)?

World Wide Web (WWW): An information space of interlinked hypertext documents and digital multimedia resources accessible over the Internet using the Hypertext Transfer Protocol (HTTP/HTTPS).
Architecture: Web pages are created using HTML, located through unique Uniform Resource Locators (URLs), and navigated across hyperlinks using a client web browser.

Q2. What is a web server? Give any two examples.

Web Server: A dedicated server application and host system that listens for client requests via HTTP/HTTPS, processes incoming requests, and delivers the requested static web files or dynamic responses back to the browser.
Examples: Apache HTTP Server, Nginx (also Microsoft IIS, LiteSpeed).

Q3. What is a CSS selector?

CSS Selector: A structural pattern declared in a Cascading Style Sheet used to target and bind style rules to specific HTML elements on a web page.
Syntax: `p { color: blue; }` (where `p` is the element selector targeting all paragraph elements).
Common Types: Element selector (e.g., `h1`), Class selector (e.g., `.intro`), and ID selector (e.g., `#header`).

Q4. Name the different methods used to specify colours in CSS.

Methods to Specify Colours in CSS:

- **Color Names:** Predefined standard keyword names (e.g., `color: red;`).
- **Hexadecimal Notation (HEX):** 6-character hex code representing RGB values (e.g., `color: #ff5733;`).
- **RGB / RGBA Functional Notation:** Integer intensities from 0 to 255 for Red, Green, and Blue, with an optional Alpha channel for opacity (e.g., `rgb(255, 0, 0);`, `rgba(0, 0, 255, 0.5);`).
- **HSL / HSLA Functional Notation:** Defines Hue (0–360°), Saturation (0–100%), Lightness (0–100%), and optional Alpha opacity (e.g., `hsl(120, 100%, 50%);`).

Q5. List the primitive data types supported by JavaScript.

Primitive Data Types in JavaScript:

- **String:** Sequence of textual characters enclosed in quotes (e.g., `"Hello"`).
- **Number:** Double-precision floating-point numbers and integers (e.g., `42`, `3.14`).
- **BigInt:** Arbitrary-precision integers for large numerical values (e.g., `9007199254740991n`).
- **Boolean:** Logical value taking either `true` or `false`.
- **Undefined:** State of a declared variable that has not been assigned a value.
- **Null:** Intentional representation of an empty or nonexistent object value.
- **Symbol:** Unique, immutable identifier used for object properties (introduced in ES6).

Q6. What is an array in JavaScript? Give an example.

Array: A specialized high-level, zero-indexed list-like object in JavaScript used to store an ordered collection of multiple values under a single variable name.
Characteristics: Arrays are dynamically resizable and can contain elements of mixed (heterogeneous) data types.
Example:

```
let languages = ["HTML", "CSS", "JavaScript"];
console.log(languages[0]); // Outputs: HTML
```

Q7. What is AJAX?

Expansion: AJAX stands for Asynchronous JavaScript and XML.
Definition: A client-side web development technique that allows web pages to send and receive data from a web server asynchronously in the background without reloading the entire page.
Core Component: Uses the browser's built-in `XMLHttpRequest` object (or modern `fetch()` API) to exchange data and dynamically update the DOM.

Q8. Name the different data types supported by PHP.

Data Types in PHP:

- **Scalar Types:** Integer (e.g., `100`), Float/Double (e.g., `19.99`), String (e.g., `"Bengaluru"`), and Boolean (`true` or `false`).
- **Compound Types:** Array (indexed or associative collections) and Object (instances of defined classes).

- **Special Types:** NULL (variable with no value) and Resource (special variable holding references to external resources, such as database links or file handlers).

Q9. What is form handling in PHP?

Form Handling in PHP: The process of capturing, validating, and processing user input submitted from an HTML `<form>` on the server side.
Mechanism: Form data submitted via the HTTP `POST` or `GET` method is collected in PHP through predefined superglobal arrays:

- `$_POST` : Collects form data sent via the HTTP POST method (secure and hidden from the URL).
- `$_GET` : Collects form data sent via the HTTP GET method (visible in the URL query string).

Syntax Example: `$name = $_POST['username'];`

Q10. What is MySQL? Expand RDBMS and SQL.

Expansions:

- **RDBMS:** Relational Database Management System.
- **SQL:** Structured Query Language.

MySQL: A popular, fast, and open-source Relational Database Management System (RDBMS) that stores data in structured tables consisting of rows and columns, utilizing SQL for data querying, manipulation, and management.
Application: Extensively utilized in server-side web development alongside PHP (LAMP/WAMP stack).

MODEL QUESTION PAPER – 3

Q1. What is client-server architecture?

Client-Server Architecture: A distributed computing structure where workloads and processing tasks are partitioned between service requesters called clients (such as web browsers) and centralized service providers called servers (such as Apache or Nginx). The client initiates communication by sending requests over a network protocol such as HTTP/HTTPS, and the server processes the application logic and delivers the requested response.

Example: A user entering a URL into Google Chrome (client), which transmits an HTTP GET request to a remote web server that returns the rendered HTML document.

Q2. What is HTML5?

HTML5: The fifth and current major version of HyperText Markup Language, standardizing the structure and presentation of World Wide Web documents. It removes dependency on proprietary third-party browser plugins by providing native multimedia embedding, standardized semantic markup, and powerful client-side scripting APIs.

Key Features: Native multimedia tags (`<video>` , `<audio>`), semantic layout elements (`<header>` , `<nav>` , `<article>` , `<footer>`), 2D drawing via `<canvas>` , and a simplified declaration: `<!DOCTYPE html>` .

Q3. What is a hyperlink? Which element is used to create it?

Hyperlink: An interactive navigation reference or clickable link in an electronic document that connects users directly to another web page, file, external website, or a specific anchor point within the same document.

Element Used: The anchor tag `<a>` is used to define hyperlinks, utilizing its mandatory `href` (hypertext reference) attribute to specify the destination resource URL.

Syntax: `<a href="https://www.example.com" target="_blank">Visit Example</a>`

Q4. Why is CSS required in web development?

CSS Requirement: Cascading Style Sheets (CSS) is required to decouple the visual presentation and layout design from the structural markup (HTML). This separation of concerns ensures that markup remains clean and semantic, while visual rules governing colors, typography, grids, and alignments are managed centrally.

Core Benefits: Enables a single external stylesheet to format hundreds of site pages consistently, supports responsive layout adjustments across varying device viewports via media queries, and speeds up page loading through browser stylesheet caching.

Q5. Write any 4 text formatting tags.

Text Formatting Tags: HTML provides inline formatting elements to modify text appearance and convey semantic meaning:

- `<strong>` : Renders bold text, indicating strong semantic importance or urgency (e.g., `<strong>Warning</strong>`).
- `<em>` : Renders italicized text, indicating stressed spoken emphasis (e.g., `<em>Note</em>`).
- `<mark>` : Renders text highlighted with a background color for reference (e.g., `<mark>Keyword</mark>`).
- `<sub>` / `<sup>` : Formats text as subscript or superscript (e.g., `H<sub>2</sub>O` , `x<sup>2</sup>`).

Q6. How to create an image in HTML?

Image Creation: An image is embedded into an HTML page using the void (self-closing) `<img>` tag. It requires the `src` attribute to provide the file path or URL of the image asset, and the `alt` attribute to supply descriptive alternate text for accessibility and screen readers.

Syntax:

```
<img src="images/banner.jpg" alt="University Campus Banner" width="600" height="250">
```

Q7. What is the conditional or ternary operator?

Conditional (Ternary) Operator: A shorthand decision-making operator (represented by the `? :` token pair) that takes three operands. It evaluates a test expression: if the condition evaluates to true, it executes and returns the first expression; otherwise, it returns the second expression.

Syntax: `condition ? expression_if_true : expression_if_false;`

Example: `let eligibility = (age >= 18) ? "Eligible" : "Not Eligible";`

Q8. What is the Document Object Model?

Document Object Model (DOM): A language-neutral application programming interface (API) that represents an HTML or XML document as a hierarchical logical tree of nodes. In this representation, each HTML element, attribute, and text block exists as an individual, programmable object.

Functionality: Allows scripting languages like JavaScript to dynamically access, add, modify, or delete HTML elements, style rules, and event listeners in real time without requiring a full page refresh.

Example: `document.getElementById("statusMsg").innerHTML = "Form Submitted Successfully";`

Q9. What is a variable in PHP? State the rules for naming PHP variables.

PHP Variable: A designated storage location in memory identified by a name, used to retain temporary values (such as strings, integers, arrays, or objects) that can be manipulated during program execution.

Naming Rules:

1. Must always start with a dollar sign prefix (`$`) followed immediately by the variable name (e.g., `$studentCount`).
2. The character following the `$` must be an alphabetical letter (`a-z` , `A-Z`) or an underscore (`_`), never a number.
3. Can contain only alphanumeric characters and underscores (`a-z` , `A-Z` , `0-9` , `_`); special symbols and whitespace are invalid.

4. Variable names are case-sensitive (e.g., `$result` and `$Result` represent two distinct variables).

Q10. What is explicit type casting in PHP?

Explicit Type Casting: The deliberate, manual conversion of a variable's data type into a specified destination type by the programmer. Unlike PHP's automatic type juggling, explicit casting forces an immediate conversion by prefixing the variable with the target type enclosed in parentheses.

Common Cast Operators: `(int)`, `(float)`, `(string)`, `(bool)`, `(array)`, and `(object)`.

Example:

```
$rawScore = "88.65";
$finalScore = (int)$rawScore; // Explicitly converted to integer: 88
```

MODEL QUESTION PAPER – 4

Q1. What is the purpose of the and <pre> elements?

** element:** A generic block-level container element used to group related HTML elements together for collective styling with CSS or dynamic manipulation using JavaScript.
<pre> element: Defines preformatted text where all spaces, tabs, and line breaks are displayed exactly as written in the source code, rendered in a monospace font.
Example:

```
<div class="card"><p>Grouped content</p></div>
<pre>Line 1    Line 2 (preserves exact spacing)</pre>
```

Q2. Which HTML element is used to insert an image? State the purposes of src and alt.

Element: The empty (void) tag `<img>` is used to embed graphic images into an HTML webpage.
src attribute: Specifies the absolute or relative file path URL of the image to be retrieved and displayed.
alt attribute: Provides alternative descriptive text displayed when the image fails to load, and read by screen readers for accessibility.
Syntax: `<img src="images/banner.jpg" alt="Website Banner" width="600" height="200">`

Q3. What is a Universal Selector?

Universal Selector: A CSS selector denoted by an asterisk (`*`) that selects and matches every single element on the web page regardless of type.
Purpose: Commonly used to apply global resets such as zeroing out default browser margins, paddings, and setting box-sizing.
Syntax:

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

Q4. What is a Grouping Selector?

Grouping Selector: A CSS technique where multiple distinct selectors that share the identical style declarations are listed together separated by commas.
Purpose: Minimizes code duplication, reduces stylesheet file size, and improves maintenance efficiency.
Syntax:

```
h1, h2, h3, p {
  font-family: Arial, sans-serif;
  color: #2c3e50;
}
```

Q5. What is an Attribute Selector?

Attribute Selector: A CSS selector that applies styling rules to HTML elements based on the presence, exact match, or partial value of their attributes, enclosed in brackets `[...]`.
Purpose: Allows targeting specific inputs or links without adding redundant class or ID names.
Syntax:

```
input[type="text"] {
  border: 1px solid #007acc;
}
a[target="_blank"] {
  color: #d9534f;
}
```

Q6. What is an event? What is an event handler?

Event: An occurrence, user interaction, or browser state change recognized by software (such as a mouse click, keypress, mouseover, or form submission).
Event Handler: A JavaScript function or block of code assigned to execute automatically in response to a triggered event.
Syntax: `<button onclick="displayAlert()">Submit</button>` or `btn.addEventListener("click", displayAlert);`

Q7. What is an expression? Name any four types of JavaScript expressions.

Expression: Any valid unit of JavaScript code that resolves, evaluates, or computes to a single resultant value.
Four Types of JavaScript Expressions:

- **Arithmetic Expressions:** Evaluates to a number (e.g., `10 + 5 * 2`).
- **String Expressions:** Evaluates to a character string (e.g., `"Hello " + "World"`).
- **Logical (Boolean) Expressions:** Evaluates to boolean true or false (e.g., `x > 10 && y < 20`).
- **Primary / Literal Expressions:** Standalone basic values and variables (e.g., `42`, `"abc"`, `true`, `null`).

Q8. What is hyperlink? How to create it?

Hyperlink: An interactive navigation element on a web page that points and connects to another web document, file, media asset, or section of the same page.
Creation: Created using the anchor element `<a>` accompanied by the mandatory `href` (Hypertext Reference) attribute containing destination URL.
Syntax: `<a href="https://bcu.ac.in" target="_blank">Visit BCU Portal</a>`

Q9. What is a session in PHP? Name any two uses of sessions.

Session: A server-side state persistence mechanism in PHP that stores user data across multiple page requests during a browser visit, tracked via a unique session ID.
Two Uses of Sessions:

- **User Authentication:** Preserving user login state and credentials across restricted dashboard pages after successful sign-in.
- **Shopping Cart Management:** Retaining selected products, quantities, and prices as the user navigates across different catalog pages.

Syntax: `session_start(); $_SESSION['username'] = 'Admin';`

Q10. What is a cookie in PHP? Name the different types of cookies.

Cookie: A small text file (up to 4 KB) created by the web server and stored on the client browser to retain user preferences and identity between visits.
Types of Cookies:

- **Session Cookies (Transient):** Stored temporarily in browser RAM and deleted automatically when the browser is closed.
- **Persistent Cookies (Permanent):** Stored on the client hard drive with a defined expiry timestamp (e.g., 30 days).

Syntax: `setcookie("user", "Nikhil", time() + (86400 * 30), "/");`

MODEL QUESTION PAPER – 5

Q1. What is a URL? Expand URL.

Expansion: URL stands for Uniform Resource Locator.
Definition: A standardized global address used to locate and identify a specific web resource (such as a webpage, document, image, or endpoint) on the Internet.
Syntax: `https://www.bcu.ac.in:443/courses/bca?sem=5` (Protocol + Hostname + Port + Path + Query String).

Q2. What is an HTML attribute? Give an example.

Definition: Special modifier keywords placed inside an element's opening tag that provide additional configuration, behavior, or properties to the HTML element.
Syntax: Written as key-value pairs in the format `name="value"`.
Example: In `<a href="https://bcu.ac.in" target="_blank">Visit BCU</a>`, `href` specifies the link URL and `target` specifies opening in a new tab.

Q3. What is HTTP? Expand HTTP.

Expansion: HTTP stands for HyperText Transfer Protocol.
Definition: An application-layer, stateless request-response communication protocol used on the World Wide Web to fetch hypermedia resources between client web browsers and web servers.
Default Ports: Port `80` for standard HTTP and Port `443` for HTTPS (secure HTTP with TLS encryption).

Q4. What is Flexbox? State its main purpose.

Definition: Flexbox (Flexible Box Layout Module) is a one-dimensional CSS3 layout model designed to distribute space and align items along a horizontal row or vertical column.
Main Purpose: To provide an efficient and responsive arrangement where container items expand or shrink to accommodate varying display sizes without using floats or positioning hacks.
Syntax: `display: flex; justify-content: center; align-items: center;`

Q5. What is CSS2? Mention any two important features.

Definition: CSS2 (Cascading Style Sheets, Level 2) is a W3C specification published in 1998 that extended CSS1 by introducing advanced visual positioning and media handling.
Two Important Features:

- **Positioning Schemes:** Introduction of absolute, relative, and fixed positioning properties alongside `z-index` layer stacking.
- **Media Types:** Support for media-specific stylesheets using rules like `@media print` and `@media screen`.

Q6. What is JSON?

Expansion: JSON stands for JavaScript Object Notation.
Definition: A lightweight, text-based, language-independent data-interchange format derived from JavaScript object syntax, used for serializing and exchanging structured data between client applications and web servers.
Syntax: `{"regNo": "U01BCA22", "name": "Rahul", "marks": 85, "passed": true}`

Q7. What is ES6? Name any four important ES6 features.

Definition: ES6 (ECMAScript 2015) is the 6th edition of the standardized ECMAScript language specification that introduced modern syntax and object-oriented capabilities to JavaScript.
Four Important Features:

1. **Block-Scoped Variables:** `let` and `const` variable declarations.
2. **Arrow Functions:** Concise function syntax with lexical `this` binding (`(x, y) => x + y`).
3. **Template Literals:** Multi-line strings and expression interpolation using backticks (`` `Hello, ${name}` ``).
4. **Classes and Modules:** Native object-oriented class syntax (`class`) with `import` and `export` statements.

Q8. Name the three types of arrays supported by PHP.

Definition: PHP supports three types of arrays to store ordered collections of multiple elements in a single variable:

- **Indexed Arrays:** Arrays with numeric index positions starting from 0 (e.g., `$colors = array("Red", "Green", "Blue");`).
- **Associative Arrays:** Arrays using named string keys associated with values (e.g., `$marks = array("Math" => 90, "CS" => 95);`).
- **Multidimensional Arrays:** Arrays containing one or more nested arrays within them (e.g., `$matrix = array(array(1, 2), array(3, 4));`).

Q9. Name the different loop statements supported by PHP.

Definition: PHP supports four iterative loop control structures to execute a block of statements repeatedly based on conditional evaluation:

1. `while` : Executes a code block as long as a specified test condition evaluates to true (entry-controlled).
2. `do...while` : Executes the code block once first, then repeats as long as the condition evaluates to true (exit-controlled).
3. `for` : Executes a block of code for a predetermined number of iterations using an initializer, condition, and counter step.
4. `foreach` : Specifically iterates through every element of an array or object collection without needing an explicit index counter.

Q10. What are CRUD operations in MySQL? Name the corresponding SQL commands.

Definition: CRUD represents the four foundational data persistence operations performed on database records in relational database systems like MySQL.
Operations & SQL Commands:

- **Create:** Inserts new records into a table → `INSERT` (e.g., `INSERT INTO users VALUES (...)`)
- **Read / Retrieve:** Queries and fetches records from a table → `SELECT` (e.g., `SELECT * FROM users`)
- **Update:** Modifies existing data records → `UPDATE` (e.g., `UPDATE users SET ... WHERE ...`)
- **Delete:** Removes unwanted records from a table → `DELETE` (e.g., `DELETE FROM users WHERE ...`)

PART II: SECTION B — 6 MARKS QUESTIONS (PAPERS 1 TO 5)

MODEL QUESTION PAPER – 1

Q11. Design a timetable using HTML tables with rowspan and colspan.

1. Core Concept

HTML tables are created using the `<table>` container tag. Rows are defined by `<tr>`, header cells by `<th>`, and data cells by `<td>`. Complex tabular structures merge adjacent cells using two key attributes:

- **rowspan:** Merges two or more vertical rows together. It specifies the number of rows a cell should span down.
- **colspan:** Merges two or more horizontal columns together. It specifies the number of columns a cell should span across.

2. Complete HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>College Class Timetable</title>
  <style>
    table { width: 100%; border-collapse: collapse; text-align: center; font-family: sans-serif; }
    th, td { border: 1px solid #333; padding: 8px; }
    th { background-color: #e2e8f0; }
    .break { background-color: #fef08a; font-weight: bold; }
    .lab { background-color: #bae6fd; font-weight: bold; }
  </style>
</head>
<body>
  <h2>BCA 5th Semester Class Timetable</h2>
  <table>
    <tr>
      <th>Day / Time</th>
      <th>09:00 - 10:00 AM</th>
      <th>10:00 - 11:00 AM</th>
      <th>11:00 - 11:30 AM</th>
      <th>11:30 - 12:30 PM</th>
      <th>12:30 - 01:30 PM</th>
    </tr>
    <tr>
      <th>Monday</th>
      <td>Data Analytics</td>
      <td>Web Programming</td>
      <td rowspan="5" class="break">T<br>E<br>A<br><br>B<br>R<br>E<br>A<br>K</td>
      <td colspan="2" class="lab">Web Programming Lab</td>
    </tr>
    <tr>
      <th>Tuesday</th>
      <td>Software Engineering</td>
      <td>Data Analytics</td>
      <td>Design Algorithms</td>
      <td>Web Programming</td>
    </tr>
    <tr>
      <th>Wednesday</th>
      <td colspan="2" class="lab">Data Analytics Lab</td>
      <td>Web Programming</td>
      <td>Software Engineering</td>
    </tr>
    <tr>
      <th>Thursday</th>
      <td>Design Algorithms</td>
      <td>Software Engineering</td>
      <td>Data Analytics</td>
      <td>Open Elective</td>
    </tr>
    <tr>
      <th>Friday</th>
      <td>Web Programming</td>
      <td>Design Algorithms</td>
      <td colspan="2" class="lab">Mini Project / Seminar</td>
    </tr>
  </table>
```

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

3. Rendered Output Representation

Day / Time	09:00 - 10:00 AM	10:00 - 11:00 AM	11:00 - 11:30 AM	11:30 - 12:30 PM	12:30 - 01:30 PM
Monday	Data Analytics	Web Programming	T E A B R E A K	Web Programming Lab (colspan="2")	
Tuesday	Software Engg	Data Analytics		Design Algorithms	Web Programming
Wednesday	Data Analytics Lab (colspan="2")			Web Programming	Software Engg
Thursday	Design Algorithms	Software Engg		Data Analytics	Open Elective
Friday	Web Programming	Design Algorithms		Mini Project / Seminar (colspan="2")	

4. Key Attributes Explanation

- **rowspan="5":** In the Monday row, the Tea Break cell spans across 5 consecutive rows vertically, keeping the break column continuous for Tuesday through Friday without repeating cells.
- **colspan="2":** Used for 2-hour lab sessions and project slots, merging two adjacent period columns horizontally.

Q12. Explain the important CSS border properties with suitable examples.

1. Core Concept

CSS border properties define the line style, thickness, and color of the boundary enclosing an HTML element. The border sits directly between the element's inner `padding` and outer `margin` in the CSS box model.

2. Key Border Properties

- **border-style:** Specifies the visual style of the border line. Required for any border to appear. Common values: `solid`, `dashed`, `dotted`, `double`, `groove`, `ridge`, `none`.
- **border-width:** Defines the thickness of the border. Values can be named (`thin`, `medium`, `thick`) or specific lengths (e.g., `2px`, `4px`).
- **border-color:** Sets the color of the border using color names, HEX codes (e.g., `#1e40af`), or RGB values (e.g., `rgb(30, 64, 175)`).
- **border (Shorthand):** Combines width, style, and color in a single declaration: `border: 2px solid #2563eb;`.
- **Individual Side Borders:** Styles specific edges independently using `border-top`, `border-right`, `border-bottom`, and `border-left`.
- **border-radius:** Defines the curvature of the element's corners to create rounded borders (e.g., `border-radius: 8px;`).

3. Working CSS Code Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .box-solid {
      border: 3px solid #2563eb;
      padding: 10px;
      margin-bottom: 10px;
    }
    .box-dashed {
      border-width: 2px;
      border-style: dashed;
      border-color: #dc2626;
      border-radius: 8px;
      padding: 10px;
      margin-bottom: 10px;
    }
    .box-custom {
      border-top: 4px solid #16a34a;
      border-bottom: 4px double #16a34a;
      border-left: none;
      border-right: none;
      padding: 10px;
    }
  </style>
</head>
<body>
  <div class="box-solid">Solid Blue Border</div>
  <div class="box-dashed">Dashed Red Border with Rounded Corners</div>
  <div class="box-custom">Custom Green Top & Bottom Borders</div>
</body>
</html>
```

4. Summary Table of Border Properties

Property	Description	Possible Values	Syntax Example
border-style	Sets the pattern of the border line.	solid, dashed, dotted, double, none	`border-style: dashed;`
border-width	Sets the thickness of the border.	thin, medium, thick, length (px, em)	`border-width: 3px;`
border-color	Sets the border color.	Color names, HEX, RGB, HSL	`border-color: #dc2626;`
border	Shorthand for width, style, and color.	[width] [style] [color]	`border: 2px solid #000;`
border-radius	Creates rounded corner edges.	length (px, em), percentage (%)	`border-radius: 10px;`

Q13. Write a JavaScript program to perform arithmetic operations using operators.

1. Core Concept

JavaScript provides standard arithmetic operators to evaluate mathematical calculations on numerical operands. These include Addition (`+`), Subtraction (`-`), Multiplication (`*`), Division (`/`), Modulus (remainder) (`%`), and Exponentiation (`**`).

2. Complete JavaScript Program

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Arithmetic Operations</title>
</head>
<body>
  <h2>JavaScript Arithmetic Calculator</h2>
  <div id="results"></div>

  <script>
    // Declare operands
    let a = 20;
    let b = 6;

    // Perform arithmetic operations
    let sum        = a + b;   // Addition
    let difference = a - b;   // Subtraction
    let product    = a * b;   // Multiplication
    let quotient   = a / b;   // Division
    let remainder  = a % b;   // Modulus (Remainder)
    let power      = a ** b;  // Exponentiation (20^6)

    // Unary increment and decrement demonstrations
    let inc = a;
    inc++; // Post-increment
    let dec = b;
    dec--; // Post-decrement

    // Format output string
    let output = "<h3>Input Values: a = " + a + ", b = " + b + "</h3>" +
      "<ul>" +
        "<li><strong>Addition (a + b):</strong> " + sum + "</li>" +
        "<li><strong>Subtraction (a - b):</strong> " + difference + "</li>" +
        "<li><strong>Multiplication (a * b):</strong> " + product + "</li>" +
        "<li><strong>Division (a / b):</strong> " + quotient.toFixed(2) + "</li>" +
        "<li><strong>Modulus (a % b):</strong> " + remainder + "</li>" +
        "<li><strong>Exponentiation (a ** b):</strong> " + power + "</li>" +
        "<li><strong>Increment (a++):</strong> " + inc + "</li>" +
        "<li><strong>Decrement (b--):</strong> " + dec + "</li>" +
      "</ul>";

    // Display result on webpage
    document.getElementById("results").innerHTML = output;

    // Log results in developer console
    console.log("Sum: " + sum);
    console.log("Quotient: " + quotient);
  </script>
</body>
</html>
```

3. Explanation of Operators Used

- **+ (Addition):** Computes the mathematical sum of `20 + 6 = 26`.
- **- (Subtraction):** Subtracts the right operand from the left: `20 - 6 = 14`.
- *** (Multiplication):** Multiplies two operands: `20 * 6 = 120`.
- **/ (Division):** Returns the floating-point quotient: `20 / 6 = 3.33`.
- **% (Modulus):** Returns the integer remainder after division: `20 % 6 = 2` (since 6 * 3 = 18, remainder 2).

- **** (Exponentiation):** Raises base `a` to the power `b`: `20 ** 6 = 64,000,000`.

4. Program Execution Output

- Addition (a + b): 26
- Subtraction (a - b): 14
- Multiplication (a * b): 120
- Division (a / b): 3.33
- Modulus (a % b): 2
- Exponentiation (a ** b): 64000000

Q14. Explain the methods used to display screen output in JavaScript.

1. Core Concept

JavaScript does not have built-in display or print functions like traditional programming languages. To present output, it interacts with browser window objects and the Document Object Model (DOM) using four primary techniques.

2. The Four Output Methods

- **1. innerHTML:** Writes dynamic content into a specified HTML element identified via the DOM (e.g., `document.getElementById()`). It interprets HTML tags within the output string and is the industry standard for UI updates without reloading the page.
- **2. document.write():** Writes raw HTML expressions directly into the document stream as the webpage parses. If invoked after the HTML document is fully loaded, it completely overwrites and deletes the existing page content.
- **3. window.alert():** Displays an interactive modal popup dialog box displaying the specified message and an OK button. It blocks browser interaction until the user acknowledges the prompt.
- **4. console.log():** Writes data directly to the browser's developer console (accessible via F12). It is primarily used for debugging, monitoring variables, and error diagnostics without disturbing the end-user display.

3. Comprehensive Code Example

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Screen Output Methods</title>
</head>
<body>
  <h2>Demonstrating JavaScript Output Methods</h2>

  <!-- Target element for innerHTML -->
  <p id="demo">Original Paragraph Content</p>

  <button onclick="showOutputs()">Execute Output Methods</button>

  <script>
    function showOutputs() {
      // Method 1: innerHTML (Recommended for DOM updates)
      document.getElementById("demo").innerHTML = "<span style='color:green;'>Updated via innerHTML!</span>";

      // Method 2: console.log() (For debugging in Developer Tools)
      console.log("Output logged to browser console successfully.");

      // Method 3: window.alert() (Modal notification dialog)
      window.alert("Alert: Operation executed!");

      // Method 4: document.write() (Direct document stream writing)
      // Note: Calling document.write() here will overwrite the loaded document
    }

    // Direct document.write() during initial page parse
    document.write("<p>Written during page parse using document.write()</p>");
  </script>
</body>
</html>
```

4. Comparison Matrix

Method	Output Target	Interprets HTML?	Primary Use Case
innerHTML	Specific DOM element	Yes	Standard dynamic webpage content updates.
document.write()	Document stream	Yes	Testing and quick prototyping during initial page load.
window.alert()	Modal dialog box	No (plain text)	Urgent user warnings, alerts, and confirmations.
console.log()	Developer console	No (text / objects)	Debugging, testing, and error inspection.

Q15. Explain how PHP works with a web server.

1. Core Concept

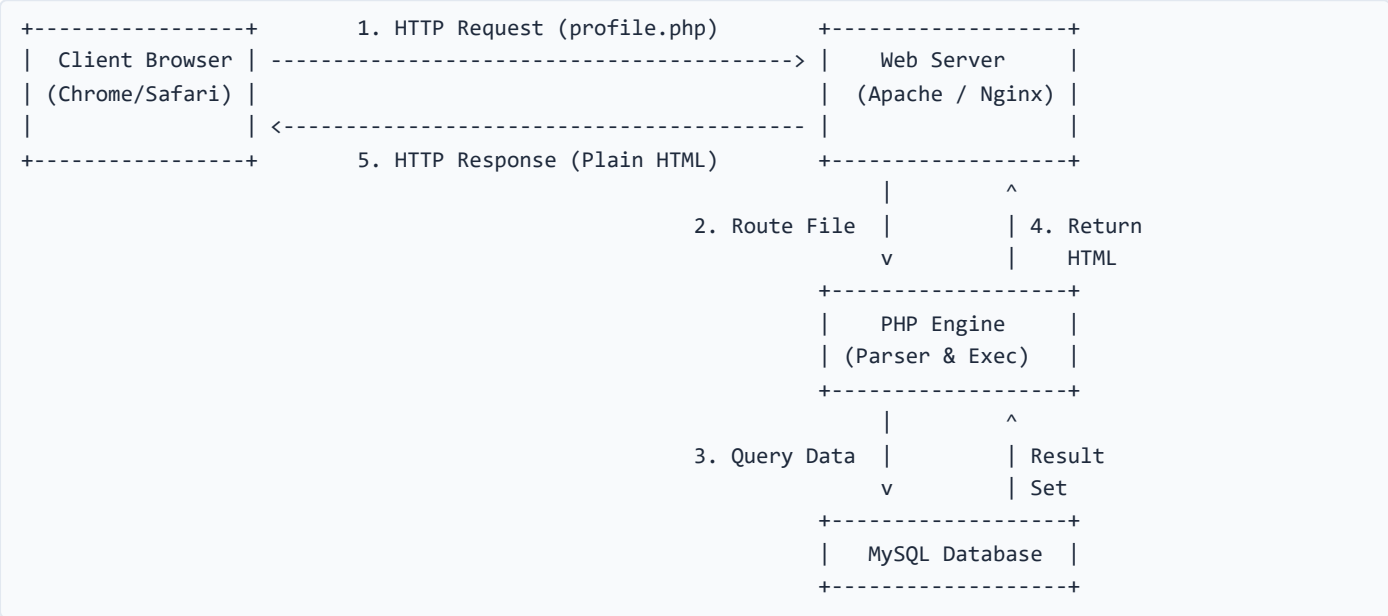
PHP is an open-source, server-side scripting language executed exclusively within the web server environment. Unlike client-side languages (JavaScript) that are downloaded and executed by the client's browser, PHP source code remains on the server. The server executes the PHP script, resolves database queries, and returns pure static HTML/CSS to the client.

2. Client-Server Request-Response Lifecycle

1. **Client Request:** The user enters a URL (e.g., `https://example.com/profile.php`) or submits a form in a web browser. The browser generates and dispatches an HTTP GET or POST request across the network to the web server.

2. **Web Server Routing:** The web server (such as Apache, Nginx, or Microsoft IIS) receives the request, examines the file extension (`.php`), and routes the request to the integrated **PHP Processing Engine** (via SAPI, mod_php, or FastCGI/PHP-FPM).
3. **PHP Execution & Database Access:** The PHP preprocessor parses the script, executes program logic, connects to a database server (such as MySQL/MariaDB) to fetch or write data, and dynamically generates HTML markup.
4. **HTML Response Generation:** The PHP engine streams the completed, generated HTML back to the web server. None of the original PHP tags (`<?php ... ?>`) or logic are sent to the client.
5. **Server Delivery & Browser Rendering:** The web server packages the HTML into an HTTP 200 response and transmits it back to the client's browser, which renders the webpage visually for the user.

3. Workflow Diagram



4. Illustrative Example

```
<!-- Source file on server: index.php -->
<!DOCTYPE html>
<html>
<body>
  <h1>Current Server Time</h1>
  <p><?php echo "Today is " . date("Y-m-d H:i:s"); ?></p>
</body>
</html>

<!-- Output received by client browser -->
<!DOCTYPE html>
<html>
<body>
  <h1>Current Server Time</h1>
  <p>Today is 2026-11-20 16:30:00</p>
</body>
</html>
```

The client receives only the resolved timestamp within plain HTML; the server-side code execution remains secure and invisible.

Q16. Explain local and global variable scope in PHP functions.

1. Core Concept

Variable scope refers to the region within a script where a variable is declared and accessible. In PHP, functions have their own isolated local scope. By default, variables defined outside functions are not accessible inside functions, and variables

created inside functions cannot be accessed outside.

2. Local Variable Scope

- **Definition:** A variable declared inside a function has local scope.
- **Accessibility:** It is accessible exclusively within that specific function block.
- **Lifespan:** It is instantiated when the function is invoked and automatically deallocated once the function completes execution.

3. Global Variable Scope

- **Definition:** A variable declared outside of any function has global scope.
- **Accessibility:** It is accessible directly anywhere outside functions.
- **Accessing Inside Functions:** To use a global variable inside a local function, PHP provides two mechanisms:
  1. **The `global` keyword:** Declaring `global $variableName;` inside the function imports the global reference.
  2. **The `$GLOBALS` superglobal array:** An associative array containing all global variables indexed by name (e.g., `$GLOBALS['x']`).

4. Complete Working PHP Code Example

```
<?php
  // Global variable declared outside functions
  $siteName = "BCA Web Portal";
  $visitorCount = 100;

  function demonstrateScope() {
    // Local variable declared inside function
    $localMsg = "Welcome, Student!";

    // Attempting to access global without declaration (Fails / Undefined)
    // echo $siteName; // Generates PHP Notice: Undefined variable

    // Accessing global variables using 'global' keyword
    global $siteName, $visitorCount;
    echo "Local Message: " . $localMsg . "<br>";
    echo "Global Site Name (via global keyword): " . $siteName . "<br>";

    // Accessing global variables using $GLOBALS superglobal array
    $GLOBALS['visitorCount']++;
    echo "Updated Visitors (via \$GLOBALS): " . $GLOBALS['visitorCount'] . "<br>";
  }

  // Execute function
  demonstrateScope();

  // Accessing global variable in script
  echo "Visitor Count outside: " . $visitorCount . "<br>";

  // Attempting to access local variable outside (Fails / Undefined)
  // echo $localMsg; // Generates PHP Notice: Undefined variable
?>
```

5. Summary Comparison

Feature	Local Variable Scope	Global Variable Scope
Declaration Location	Inside a function block.	Outside of any function.
Access Inside Functions	Directly accessible.	Requires `global` keyword or `$GLOBALS['var']`.
Access Outside Functions	Inaccessible (generates undefined warning).	Directly accessible.
Memory Lifetime	Created upon function call, destroyed on return.	Persists throughout the entire script execution.

MODEL QUESTION PAPER – 2

Q11. Create ordered, unordered, and definition lists for course details.

1. Core Concept:

HTML provides three distinct list elements to structure items hierarchically based on relationships and numbering requirements:

- **Ordered List (`<ol>`):** Renders items in a sequential, numerical, or alphabetical order using list item (`<li>`) tags.
- **Unordered List (`<ul>`):** Displays items as a bulleted collection where sequential order does not signify priority.
- **Definition / Description List (`<dl>`):** Organizes terms (`<dt>`) paired with their corresponding descriptions (`<dd>`), ideal for course glossaries and syllabi.

2. Working HTML Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Course Details Lists</title>
</head>
<body>
  <h2>BCA Course Curriculum Details</h2>

  <!-- 1. Ordered List: Semester Sequence -->
  <h3>1. Semester Progression (Ordered List)</h3>
  <ol type="1" start="1">
    <li>Semester I: Problem Solving with C</li>
    <li>Semester II: Object Oriented Programming with Java</li>
    <li>Semester III: Database Management Systems</li>
    <li>Semester IV: Python Programming</li>
    <li>Semester V: Web Programming &amp;amp; Data Analytics</li>
  </ol>

  <!-- 2. Unordered List: Elective Subjects -->
  <h3>2. Elective Subjects Available (Unordered List)</h3>
  <ul style="list-style-type: square;">
    <li>Cloud Computing &amp;amp; DevOps</li>
    <li>Artificial Intelligence &amp;amp; Machine Learning</li>
    <li>Cyber Security &amp;amp; Ethical Hacking</li>
    <li>Mobile Application Development</li>
  </ul>

  <!-- 3. Definition List: Course Terminology -->
  <h3>3. Course Terminology Glossary (Definition List)</h3>
  <dl>
    <dt><strong>DCCA511:</strong></dt>
    <dd>Core course code representing Web Programming covering HTML, CSS, JavaScript, PHP, and MySQL.</dd>
    <dt><strong>Credits:</strong></dt>
    <dd>Quantitative weight assigned to the course denoting 4 lecture hours per week.</dd>
    <dt><strong>CIE:</strong></dt>
    <dd>Continuous Internal Evaluation comprising tests, assignments, and practical laboratory tests.</dd>
  </dl>
</body>
</html>
```

3. List Elements and Attributes Summary:

List Type	Container Tag	Item Tag	Key Attributes / CSS Styling
Ordered List	`<ol>`	`<li>`	`type="1|a|A|i|I"`, `start="n"`, `reversed`
Unordered List	`<ul>`	`<li>`	`list-style-type: disc | circle | square | none;`
Definition List	`<dl>`	`<dt>`, `<dd>`	No counter attributes; supports semantic term-definition pairs

4. Expected Visual Output:

1. Semester Progression (Ordered List)

1. Semester I: Problem Solving with C
2. Semester II: Object Oriented Programming with Java
3. Semester III: Database Management Systems

...

2. Elective Subjects Available (Unordered List)

■ Cloud Computing & DevOps
■ Artificial Intelligence & Machine Learning
■ Cyber Security & Ethical Hacking

3. Course Terminology Glossary (Definition List)
DCCA511:
Core course code representing Web Programming covering HTML, CSS, JavaScript, PHP, and MySQL.
Credits:
Quantitative weight assigned to the course denoting 4 lecture hours per week.

Q12. Explain the important functions of a web browser.

1. Core Concept:

A **Web Browser** is a client application software that locates, retrieves, renders, and displays content from the World Wide Web. It translates server responses (HTML, CSS, JavaScript, images) into interactive, human-readable visual web pages.

2. Browser Operational Workflow:

```
[User enters URL]
      |
      ▼
[DNS Lookup: Resolves Domain to IP]
      |
      ▼
[HTTP / HTTPS Request to Web Server]
      |
      ▼
[Server returns HTML, CSS, JS payloads]
      |
      ▼
[Browser Parsing Engine builds DOM + CSSOM Trees]
      |
      ▼
[Render Tree Construction -> Layout (Reflow) -> Painting]
      |
      ▼
[Interactive Page Rendered on User Screen]
```

3. Key Functions of a Web Browser:

- **1. Requesting and Fetching Resources:** Sends client HTTP/HTTPS requests to remote web servers based on the URL entered or hyperlink clicked, and receives incoming payload packets.
- **2. HTML and CSS Parsing (Rendering Engine):** The layout engine (e.g., Blink, WebKit, Gecko) parses HTML tags to build the Document Object Model (DOM) tree and parses CSS stylesheets to build the CSS Object Model (CSSOM) tree.
- **3. Layout and Painting:** Combines the DOM and CSSOM into a Render Tree, computes the exact geometric dimensions and coordinates of every element (layout/reflow), and paints visual pixels onto the display screen.
- **4. JavaScript Execution (Scripting Engine):** Executes client-side JavaScript via integrated engines (e.g., Google V8 in Chrome, SpiderMonkey in Firefox) to handle user events, validate forms, and manipulate the DOM dynamically.
- **5. Navigation and History Management:** Provides controls including Back, Forward, Refresh, Stop, Home, and maintains browsing session history and tabbed browsing instances.
- **6. Security, Privacy, and State Storage:** Validates SSL/TLS digital security certificates, enforces Same-Origin Policy (SOP), manages cookies, local storage (`localStorage` , `sessionStorage`), and caches static assets to accelerate repeat visits.

4. Concrete Example:

When a user navigates to an online result portal, the browser resolves the domain, retrieves HTML and CSS, renders the grade table, executes client-side JavaScript to validate the student registration number, and securely caches stylesheets for subsequent page views.

Q13. Design a webpage using Inline, Internal, and External CSS.

1. Core Concept:

Cascading Style Sheets (CSS) format HTML elements via three integration approaches:

- **Inline CSS:** Applied directly inside an individual HTML element using the `style` attribute (highest specificity).
- **Internal (Embedded) CSS:** Declared inside the `<style>` tag within the `<head>` section of the HTML document.
- **External CSS:** Defined in an independent `.css` file and linked using the `<link>` element in the `<head>` section.

2. Complete Working Code:

File 1: `styles.css` (External Stylesheet):

```
/* External stylesheet defining global page aesthetics */
body {
  font-family: Arial, sans-serif;
  background-color: #f4f6f8;
  margin: 20px;
}

.site-header {
  color: #1e3a8a;
  text-align: center;
  border-bottom: 2px solid #1e3a8a;
  padding-bottom: 8px;
}

.external-box {
  background-color: #ffffff;
  border: 1px solid #cbd5e1;
  padding: 15px;
  border-radius: 6px;
  margin-bottom: 15px;
}
```

File 2: `index.html` (HTML Webpage Integrating All Three CSS Types):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Integration Demonstration</title>

  <!-- 1. EXTERNAL CSS: Linked from styles.css -->
  <link rel="stylesheet" href="styles.css">

  <!-- 2. INTERNAL CSS: Scoped within the <style> block -->
  <style>
    .internal-card {
      background-color: #e0f2fe;
      border: 1.5px solid #0284c7;
      padding: 14px;
      margin: 12px 0;
      border-radius: 4px;
    }
    .internal-title {
      color: #0369a1;
      font-size: 17px;
      margin-top: 0;
    }
  </style>
</head>
<body>

  <!-- Styled via External CSS -->
  <h1 class="site-header">Web Programming CSS Showcase</h1>

  <div class="external-box">
    <p>This container is styled via <code>External CSS</code> linked from <code>styles.css</code>.</p>
  </div>

  <!-- Styled via Internal CSS -->
  <div class="internal-card">
    <h3 class="internal-title">Internal CSS Block</h3>
    <p>This section is formatted using an embedded <code>&lt;style&gt;</code> tag in the head.</p>
  </div>

  <!-- Styled via Inline CSS -->
  <div style="background-color: #fef08a; border: 1.5px solid #ca8a04; padding: 12px; border-radius: 4px;">
    <h3 style="color: #854d0e; margin: 0 0 6px 0;">Inline CSS Demonstration</h3>
    <p style="color: #713f12; font-weight: bold; margin: 0;">
      This paragraph is styled directly using the <code>style=""</code> attribute on individual HTML tags.
    </p>
  </div>

</body>
</html>
```

3. Comparison and Cascading Precedence:

CSS Type	Location	Scope	Cascading Priority
Inline CSS	Inside element `style="..."`	Single element only	Highest Priority (Overrides Internal & External)
Internal CSS	Inside `<style>` in `<head>`	Single HTML document	Medium Priority (Overrides External if placed after)
External CSS	Separate `.css` file via `<link>`	Entire website across multiple pages	Standard Priority (Overridden by Internal & Inline)

4. Expected Output:

The webpage renders a centered dark blue heading and white card from the external stylesheet, an sky-blue box with dark blue text from the internal stylesheet, and a highlighted amber alert box from the inline styles.

Q14. Explain the important CSS text properties with suitable examples.

1. Core Concept:

CSS text properties control the visual presentation, spacing, alignment, decoration, and capitalization of textual elements, ensuring legible and structured typographic design across web browsers.

2. Full Syntax & Working CSS Code Example:

```
/* Typographic styling applying core CSS text properties */
.article-header {
  color: #1e293b;                 /* Foreground text color */
  text-align: center;             /* Horizontal alignment */
  text-transform: uppercase;      /* Capitalization transform */
  letter-spacing: 2px;            /* Character spacing */
  text-shadow: 1px 1px 2px #94a3b8;/* Drop shadow */
}

.article-body {
  color: #334155;
  text-align: justify;            /* Justified paragraph alignment */
  text-indent: 30px;              /* First line indent */
  line-height: 1.8;               /* Interline vertical spacing */
  word-spacing: 3px;              /* Spacing between individual words */
}

.highlighted-link {
  text-decoration: underline;     /* Line decoration */
  text-decoration-color: #ef4444; /* Decoration line color */
}
```

3. Important CSS Text Properties Explanation:

CSS Property	Possible Values	Function & Description	Example Syntax
`color`	Color names, HEX, RGB, HSL	Sets the foreground color of textual characters.	`color: #2563eb;`
`text-align`	`left`, `right`, `center`, `justify`	Aligns text horizontally within its parent block container.	`text-align: center;`
`text-decoration`	`none`, `underline`, `overline`, `line-through`	Adds or removes decorative lines (e.g., removing link underlines).	`text-decoration: none;`
`text-transform`	`uppercase`, `lowercase`, `capitalize`, `none`	Alters the capitalization of text without altering HTML markup.	`text-transform: uppercase;`
`text-indent`	Length in `px`, `em`, `%`	Indents the first line of text in a block element.	`text-indent: 25px;`
`line-height`	Numbers (e.g., `1.6`), `px`, `%`	Sets the vertical distance between baselines of text lines.	`line-height: 1.6;`
`letter-spacing`	Length in `px`, `em` (positive or negative)	Adjusts horizontal tracking space between individual characters.	`letter-spacing: 1.5px;`
`word-spacing`	Length in `px`, `em`	Adjusts horizontal white-space distance between separate words.	`word-spacing: 4px;`
`text-shadow`	`x-offset y-offset blur color`	Attaches a drop-shadow effect to textual characters.	`text-shadow: 2px 2px 4px #888;`

4. Concrete Usage:

Web articles combine `text-align: justify` and `line-height: 1.7` to maximize readability on mobile and desktop screens, while navigation links use `text-decoration: none` and `text-transform: uppercase` for clean menu bars.

Q15. Explain the length property and important array methods in JavaScript.

1. Core Concept:

In JavaScript, an **Array** is an ordered, zero-indexed collection of values. The `length` property reflects the total number of elements present. JavaScript provides built-in prototype methods to insert, remove, extract, search, and iterate over array elements.

2. Full Working JavaScript Code Example:

```
// 1. Initializing an array and checking length
let subjects = ["HTML", "CSS", "JavaScript"];
console.log("Initial Length:", subjects.length); // 3

// 2. Adding and removing elements at ends
subjects.push("PHP");              // Adds to the end -> ["HTML", "CSS", "JavaScript", "PHP"]
let lastItem = subjects.pop();     // Removes from end -> returns "PHP"

subjects.unshift("Bootstrap");     // Adds to the beginning -> ["Bootstrap", "HTML", "CSS", "JavaScript"]
let firstItem = subjects.shift();   // Removes from beginning -> returns "Bootstrap"

// 3. Modifying with splice() and extracting with slice()
// splice(startIndex, deleteCount, itemToAdd)
subjects.splice(1, 0, "XML");      // Inserts "XML" at index 1 without deleting

// slice(startIndex, endIndex) - non-mutating
let webBasics = subjects.slice(0, 2); // Extracts indices 0 and 1

// 4. Joining and searching
let subjectString = subjects.join(", "); // Returns "HTML, XML, CSS, JavaScript"
let hasJS = subjects.includes("JavaScript"); // Returns true
let cssIndex = subjects.indexOf("CSS");      // Returns 2

console.log("Updated Array:", subjects);
console.log("Array as String:", subjectString);
console.log("Updated Length:", subjects.length);
```

3. Detailed Explanation of Array Methods:

Method / Property	Category	Behavior & Description	Mutates Original Array?
`length`	Property	Returns the total count of elements; setting it can truncate the array.	Yes (if assigned)
`push(item)`	Insertion	Appends one or more items to the end of the array; returns new length.	Yes
`pop()`	Deletion	Removes and returns the last element of the array.	Yes
`unshift(item)`	Insertion	Inserts one or more items at the beginning; returns new length.	Yes
`shift()`	Deletion	Removes and returns the first element of the array.	Yes
`splice(start, del, add)`	Modification	Adds, removes, or replaces elements at any arbitrary index.	Yes
`slice(start, end)`	Extraction	Returns a shallow copy of a portion of an array into a new array.	No
`concat(arr2)`	Combination	Merges two or more arrays and returns the new combined array.	No
`join(separator)`	String Conversion	Concatenates all array elements into a string separated by a delimiter.	No
`indexOf(item)`	Search	Returns the first index of an element, or `-1` if not found.	No

4. Console Output:

Initial Length: 3
Updated Array: ['HTML', 'XML', 'CSS', 'JavaScript']

Array as String: HTML, XML, CSS, JavaScript
Updated Length: 4

Q16. Explain the GET method and the use of the $_GET superglobal array in PHP.

1. Core Concept:

In HTTP communication, the **GET Method** requests data from a specified resource by appending form inputs directly to the Uniform Resource Locator (URL) as visible key-value pairs (query string). In PHP, the predefined `$_GET` superglobal array automatically captures, parses, and provides programmatic access to these query parameters.

2. Full Working Code Example:

File 1: `search.html` (HTML Client Form):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Student Course Search</title>
</head>
<body>
  <h2>Course Search Portal</h2>
  <!-- method="GET" appends form inputs to the destination URL -->
  <form action="search_results.php" method="GET">
    <label for="st_name">Student Name:</label><br>
    <input type="text" id="st_name" name="student_name" required><br><br>

    <label for="crs">Course:</label><br>
    <input type="text" id="crs" name="course_name" required><br><br>

    <input type="submit" value="Search Course">
  </form>
</body>
</html>
```

File 2: `search_results.php` (PHP Server Script Using `$_GET`):

```
<?php
// Check if the expected GET parameters are present in the request
if (isset($_GET['student_name']) && isset($_GET['course_name'])) {

    // Sanitize user input using htmlspecialchars to prevent XSS injection
    $student = htmlspecialchars($_GET['student_name']);
    $course  = htmlspecialchars($_GET['course_name']);

    echo "<h3>Search Query Processed Successfully</h3>";
    echo "<p>Student Name: <strong>" . $student . "</strong></p>";
    echo "<p>Course Enrolled: <strong>" . $course . "</strong></p>";

} else {
    echo "<p style='color: red;'>Error: No search parameters provided via GET query string.</p>";
}
?>
```

3. Key Characteristics and Working Mechanism of `$_GET`:

- **Superglobal Scope:** `$_GET` is accessible everywhere within PHP scripts (inside functions, methods, or files) without requiring a `global $variable;` declaration.
- **URL Query String Format:** Form inputs are converted into URL format: `search_results.php?student_name=Nikhil&course_name=BCA`. The `?` marks the start of the query string and `&` separates multiple key-value parameters.

- **Visibility and Bookmarking:** Because parameter values appear in the browser address bar, users can bookmark URLs with pre-populated parameters (e.g., search engines, pagination links).
- **Data Length Constraints:** Constrained by web server and browser maximum URL limits (typically around 2048 characters).
- **Security Rule:** `GET` must never be used for sensitive information (e.g., passwords, credit card numbers) because values are stored in browser history, web server access logs, and proxy caches.

4. Execution Output:

URL Produced: `http://localhost/search_results.php?student_name=Nikhil&course_name=BCA`

Rendered Webpage Output:
Search Query Processed Successfully
Student Name: **Nikhil**
Course Enrolled: **BCA**

MODEL QUESTION PAPER – 3

Q11. Develop a student registration form using HTML form controls.

1. Core Concept and Form Controls Overview

Definition: An HTML form is an interactive document component defined by the `<form>` element, used to collect user inputs and submit them to a server-side processing script via HTTP methods like POST or GET. A comprehensive student registration form incorporates several diverse input controls:

- `<input type="text">` : Single-line field for names and enrollment numbers.
- `<input type="email">` : Validates email address format natively in the browser.
- `<input type="password">` : Masks entered characters for account security.
- `<input type="radio">` : Mutually exclusive selection for gender.
- `<input type="checkbox">` : Multi-selection option for student hobbies or extracurriculars.
- `<select>` and `<option>` : Dropdown list for selecting degree program or department.
- `<textarea>` : Multi-line text field for entering physical residential address.
- `<input type="submit">` and `<input type="reset">` : Action buttons to send or clear data.

2. Complete Working HTML Form Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Student Registration Form</title>
</head>
<body>
  <h2>Student Registration Form</h2>
  <form action="register.php" method="POST">
    <!-- Text Input: Student Name -->
    <label for="name">Full Name:</label><br>
    <input type="text" id="name" name="student_name" placeholder="Enter full name" required><br><br>

    <!-- Email Input -->
    <label for="email">Email Address:</label><br>
    <input type="email" id="email" name="student_email" placeholder="student@example.com" required><br><br>

    <!-- Password Input -->
    <label for="pwd">Password:</label><br>
    <input type="password" id="pwd" name="student_pwd" required><br><br>

    <!-- Radio Buttons: Gender -->
    <label>Gender:</label><br>
    <input type="radio" id="male" name="gender" value="Male" required>
    <label for="male">Male</label>
    <input type="radio" id="female" name="gender" value="Female">
    <label for="female">Female</label>
    <input type="radio" id="other" name="gender" value="Other">
    <label for="other">Other</label><br><br>

    <!-- Dropdown Select: Course -->
    <label for="course">Select Course:</label><br>
    <select id="course" name="course" required>
      <option value="">-- Select Department --</option>
      <option value="BCA">Bachelor of Computer Applications (BCA)</option>
      <option value="BSc">B.Sc. Computer Science</option>
      <option value="BTech">B.Tech Information Technology</option>
    </select><br><br>

    <!-- Checkboxes: Interests -->
    <label>Extracurricular Interests:</label><br>
    <input type="checkbox" id="coding" name="interests[]" value="Coding">
    <label for="coding">Coding Club</label>
    <input type="checkbox" id="sports" name="interests[]" value="Sports">
    <label for="sports">Sports</label>
    <input type="checkbox" id="music" name="interests[]" value="Music">
    <label for="music">Music</label><br><br>

    <!-- Textarea: Permanent Address -->
    <label for="address">Permanent Address:</label><br>
    <textarea id="address" name="address" rows="4" cols="35" placeholder="Enter postal address"></textarea><br

    <!-- Form Submission Buttons -->
    <input type="submit" value="Register Student">
    <input type="reset" value="Clear Form">
  </form>
</body>
</html>
```

3. Key Form Attributes and Control Roles

- **action:** Identifies the destination server endpoint (`register.php`) that handles submitted parameters.
- **method:** Set to `POST` so that sensitive payload data is transmitted in the HTTP message body rather than appended visibly to the URL.
- **name attribute:** Serves as the associative key by which server-side scripts identify each submitted value.
- **required attribute:** Triggers native HTML5 client-side validation to block form submission if blank.

Q12. Write a JavaScript program to determine whether a number is even or odd using conditional statements.

1. Concept and Arithmetic Logic

Definition: In mathematics, an integer is classified as even if it is completely divisible by 2 with a remainder of zero; otherwise, it is classified as odd. In JavaScript, the arithmetic modulus operator (`%`) calculates the remainder of integer division. The decision is made using an `if...else` conditional statement that tests if `number % 2 === 0`.

2. Full Working JavaScript Program

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Even or Odd Checker</title>
  <script>
    function checkEvenOrOdd() {
      // Step 1: Read input value from the text field
      var inputElement = document.getElementById("numInput");
      var rawValue = inputElement.value;

      // Step 2: Validate non-empty and numeric input
      if (rawValue.trim() === "" || isNaN(rawValue)) {
        document.getElementById("result").innerText = "Please enter a valid numeric integer.";
        return;
      }

      // Step 3: Convert string input to integer
      var num = parseInt(rawValue, 10);

      // Step 4: Evaluate condition using modulus operator (%)
      if (num % 2 === 0) {
        document.getElementById("result").innerText = num + " is an EVEN number.";
      } else {
        document.getElementById("result").innerText = num + " is an ODD number.";
      }
    }
  </script>
</head>
<body>
  <h3>JavaScript Even or Odd Number Checker</h3>
  <label for="numInput">Enter an integer:</label>
  <input type="text" id="numInput" placeholder="e.g. 14">
  <button onclick="checkEvenOrOdd()">Check Number</button>
  <p id="result" style="font-weight: 600;"></p>
</body>
</html>
```

3. Program Logic Breakdown

- **parseInt(rawValue, 10):** Converts user string input into a base-10 signed integer.
- **isNaN(rawValue):** Guards against non-numeric characters before executing calculations.

- **num % 2 === 0:** Computes the remainder of `num / 2`. When the remainder equals 0, the test branch evaluates to `true`.
- **document.getElementById():** Interacts with the DOM tree to update the output paragraph dynamically without page reload.

4. Sample Output Runs

- **Input:** `18` → **Output:** `18 is an EVEN number.`
- **Input:** `25` → **Output:** `25 is an ODD number.`
- **Input:** `0` → **Output:** `0 is an EVEN number.`

Q13. Explain font-family and the different generic font families.

1. Concept of font-family and the Font Stack

Definition: In CSS, the `font-family` property specifies a prioritized list of typeface family names and generic fallback families for an element. Because target devices may not have specific commercial fonts installed, web developers define a font stack separated by commas. The browser parses the list sequentially from left to right, applying the first locally available font, and defaulting to a universal generic family at the end of the stack.

Syntax Rule: Specific font names containing spaces (such as `"Times New Roman"` or `"Courier New"`) must be enclosed in quotes, while generic font family keywords do not use quotes.

2. The Five Standard Generic Font Families

Generic Family	Typographic Characteristics	Common Specific Fonts	Typical Usage
`serif`	Characters have small decorative projections, hooks, or horizontal finishing strokes (serifs) at stroke terminations.	Times New Roman, Georgia, Garamond	Formal print media, editorial articles, books, academic papers.
`sans-serif`	Characters have clean, stroke endings with no decorative terminal projections (without serifs).	Arial, Helvetica, Verdana, Tahoma, Trebuchet MS	Digital computer screens, UI elements, mobile applications, modern web pages.
`monospace`	Every individual glyph occupies the exact same horizontal character width (fixed-width typography).	Courier New, Consolas, Monaco, Lucida Console	Programming code snippets, terminal consoles, aligned tabular data.
`cursive`	Glyphs feature flowing, joined or cursive strokes mimicking human handwriting and calligraphy.	Comic Sans MS, Brush Script MT, Lucida Handwriting	Informal notes, decorative headlines, invitations, certificate signatures.
`fantasy`	Highly stylized, decorative, ornamental letterforms designed for artistic impact.	Impact, Papyrus, Copperplate	Artistic poster headers, video game titles, promotional banners.

3. CSS Implementation Example

```
/* Example of font-family stacks with generic fallbacks */
body {
  /* Sans-serif stack: modern, legible screen typography */
  font-family: "Segoe UI", Arial, Helvetica, sans-serif;
}

h1, h2 {
  /* Serif stack: traditional, elegant headers */
  font-family: "Georgia", "Times New Roman", serif;
}

pre, code {
  /* Monospace stack: fixed-width character alignment */
  font-family: "Consolas", "Courier New", monospace;
}
```

Q14. Explain CSS layout and the role of the display property.

1. Core Concept of CSS Layout

Definition: CSS Layout determines the spatial arrangement, sizing, flow, and visual positioning of HTML elements across the browser viewport. By default, browsers arrange HTML elements according to normal document flow based on their intrinsic display type. The `display` property is the foundational CSS property used to control an element's inner and outer display box behavior, dictating whether it creates a new line or sits inline with surrounding content.

2. Core Values of the Display Property

- **display: block:** The element begins on a new line and stretches horizontally to occupy the entire available width of its parent container. Explicit `width`, `height`, `margin`, and `padding` properties are respected on all four sides. Common default block tags: `<div>`, `<p>`, `<h1>-<h6>`, `<section>`.
- **display: inline:** The element flows continuously within the text stream on the same line, taking up only as much width as its enclosed content requires. It does not start on a new line. Top and bottom margins and explicit `width` / `height` declarations have no effect. Common default inline tags: `<span>`, `<a>`, `<strong>`, `<em>`.
- **display: inline-block:** A hybrid display mode that places the element inline with adjacent text on the same row, while behaving internally as a block-level container. This allows developers to set explicit `width`, `height`, and vertical `margin` / `padding` values. Highly utilized for navigation links and buttons.
- **display: none:** Completely excises the element from the document layout flow. The element becomes invisible and occupies zero dimensional space on screen (distinct from `visibility: hidden`, which hides the element while reserving its layout space).

3. Modern Advanced Layout Modes

- **display: flex:** Establishes a one-dimensional flexible box container, allowing dynamic redistribution of space and alignment of items along a single main axis (row or column).
- **display: grid:** Establishes a two-dimensional grid container structured with simultaneous columns and rows for complex page scaffolding.

4. Code Example Demonstrating Display Roles

```
/* Converting inline links into inline-block clickable buttons */
.nav-link {
  display: inline-block;
  width: 120px;
  height: 36px;
  line-height: 36px;
  text-align: center;
  margin: 0 8px;
  background-color: #0f172a;
  color: #ffffff;
  text-decoration: none;
}

/* Modal dialog hidden by default */
.modal-window {
  display: none;
}
```

Q15. Explain the POST method and the use of the $_POST superglobal array in PHP.

1. Concept of the HTTP POST Method

Definition: The HTTP POST method is a protocol request mechanism used to transmit form data from a client browser to a web server within the payload body of the HTTP request, rather than appending parameters to the URL query string. Because values are packed into the message body, POST is the required method for sensitive transactions (passwords, payment information) and high-volume data transmissions (file uploads, long text).

2. The $_POST Superglobal Array in PHP

Definition: `$_POST` is an automatic, predefined superglobal associative array in PHP. It automatically gathers and stores all key-value pairs submitted to the current script via the HTTP POST method.

- **Global Scope:** As a superglobal, `$_POST` is accessible anywhere in a PHP script (inside functions, classes, or included files) without declaring `global $_POST;`.
- **Key Association:** Keys in the `$_POST` array correspond directly to the `name` attributes assigned to the originating HTML form controls (e.g., `$_POST['username']`).

3. Comparison: POST Method vs GET Method

Feature	POST Method ($_POST)	GET Method ($_GET)
Data Visibility	Data is enclosed in the HTTP request body; completely hidden from URL bar.	Data is appended visibly to the URL query string (`?key=value`).
Security	Suitable for passwords, authentication tokens, and private data.	Insecure for sensitive data; values are stored in browser history and server logs.
Payload Capacity	Effectively unlimited (governed by `post_max_size` in `php.ini`).	Limited by browser URL length constraints (approximately 2,048 characters).
Idempotency	Non-idempotent; used for actions that change server state (inserting/updating records).	Idempotent; used for safe queries, searches, and bookmarkable pages.

4. Full Implementation Example

```
<!-- HTML Form (login.html) -->
<form action="login.php" method="POST">
  <label>Username:</label>
  <input type="text" name="uname" required><br>
  <label>Password:</label>
  <input type="password" name="pword" required><br>
  <input type="submit" value="Log In">
</form>

<?php
// PHP Script (login.php)
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Check if keys exist in $_POST array
    if (isset($_POST["uname"]) && isset($_POST["pword"])) {
        // Sanitize and read input values
        $username = htmlspecialchars($_POST["uname"]);
        $password = $_POST["pword"];

        echo "Login attempt received for user: " . $username;
    }
}
?>
```

Q16. What is MySQL? Write the features of MySQL.

1. Core Concept and Definition

Definition: MySQL is an open-source, high-performance Relational Database Management System (RDBMS) based on Structured Query Language (SQL). It organizes and stores data in structured tabular relations comprising rows (records) and columns (attributes), enforcing relational data integrity through primary keys, foreign keys, and constraints. Developed originally by MySQL AB and currently maintained by Oracle Corporation, it serves as the foundational database layer for the LAMP/WAMP web development stack.

2. Major Features of MySQL

- **Relational Database Model:** Organizes data into normalized tables with strict schema definitions, facilitating relationships (one-to-one, one-to-many, many-to-many) and preventing data duplication.
- **ACID Compliance:** Through its default storage engine (InnoDB), MySQL supports transactions that guarantee Atomicity, Consistency, Isolation, and Durability, ensuring reliable commit/rollback handling.
- **High Speed and Performance:** Employs memory caching, cost-based query optimization, and multiple indexing strategies (B-tree, hash indexes) to execute high-throughput read and write operations.
- **Cross-Platform Portability:** Runs natively across major operating systems including Microsoft Windows, Linux, macOS, and Unix distributions.
- **Multi-Threading and Multi-User Concurrency:** Features a fully multi-threaded architecture capable of concurrently serving thousands of simultaneous connections with row-level locking to minimize contention.
- **Robust Security Architecture:** Incorporates host-based user access privileges, encrypted data transfer using SSL/TLS, and hashed password authentication.
- **Pluggable Storage Engine Architecture:** Supports interchangeable storage engines tailored for specific workloads: InnoDB (transactional), MyISAM (read-intensive), and Memory (ultra-fast volatile caching).
- **Seamless Web Stack Integration:** Offers native client libraries and connectors for PHP (`mysqli` , `PDO`), Python, Java, and C++, making it the industry standard for database-driven websites.

3. Relational SQL Demonstration

```
-- Creating a normalized relational table in MySQL
CREATE TABLE Students (
    StudentID INT AUTO_INCREMENT PRIMARY KEY,
    FullName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    EnrolledCourse VARCHAR(50) DEFAULT 'BCA',
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;

-- Inserting a record
INSERT INTO Students (FullName, Email, EnrolledCourse)
VALUES ('Nikhil P', 'nikhil@example.com', 'BCA');
```

MODEL QUESTION PAPER – 4

Q11. Differentiate between Internet and WWW.

1. Core Concept and Definitions

Internet: A massive global network of interconnected computer hardware, cables, switches, and routing devices communicating through standardized protocol suites, primarily TCP/IP. It constitutes the physical telecommunication highway enabling worldwide resource exchange.
World Wide Web (WWW): An information retrieval service operating on top of the Internet infrastructure. It consists of interlinked hypertext documents, multimedia resources, and web applications accessible through web browsers using the HTTP/HTTPS protocol.

2. Architecture and Working Analogy

Analogy: The Internet is analogous to a global system of roadways, tracks, and physical bridges, while the World Wide Web represents the vehicles, cargo, and passenger services traveling across those roads.

3. Comprehensive Comparison Table

Comparison Parameter	The Internet	World Wide Web (WWW)
Definition	A global physical network of interconnected computers and networking hardware.	A collection of interlinked multimedia documents and software services on the Internet.
Origin and Inventor	Originated in 1969 as ARPANET; developed by Vinton Cerf and Robert Kahn.	Invented in 1989 by Sir Tim Berners-Lee at CERN laboratories in Geneva.
Nature	Hardware-centric network infrastructure (telecommunication backbone).	Software-centric information layer (application service).
Primary Protocols	TCP, IP, UDP, ICMP, ARP, BGP.	HTTP, HTTPS, WebSocket.
Addressing System	IP Addresses (IPv4: 32-bit, IPv6: 128-bit).	Uniform Resource Locators (URLs) and URIs.
Access Mechanism	Networking hardware, network interfaces, routers, and gateways.	Web browser software (Google Chrome, Mozilla Firefox, Safari).
Scope of Services	Encompasses WWW, Email (SMTP), File Transfer (FTP), Telnet, and VoIP.	A subset service of the Internet focused on websites, HTML, and web apps.
Mutual Dependency	Can operate independently without the existence of the World Wide Web.	Cannot exist or function without the underlying Internet infrastructure.

4. Practical Example

Example: Sending an email through a desktop mail client using SMTP and IMAP transfers data across the Internet without utilizing the World Wide Web. Conversely, reading email through a web browser using `https://mail.google.com` uses both the Internet and the World Wide Web.

Q12. Explain headings, paragraphs, and line breaks in HTML with suitable examples.

1. Core Concept and Element Definitions

Headings (<h1> to <h6>): Block-level elements used to define hierarchical titles and structural divisions within an HTML document. `<h1>` represents the most significant primary heading, while `<h6>` represents the least significant heading. Browsers render headings in bold with descending font sizes.
Paragraphs (<p>): Block-level elements used to wrap and structure continuous blocks of textual prose. Browsers automatically insert vertical whitespace margins before and after each paragraph.
Line Breaks (<br>): An empty (void) inline tag used to insert a single carriage-return line break within text without creating extra paragraph spacing. It does not require a closing tag.

2. Full Working HTML Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Headings, Paragraphs, and Line Breaks Demo</title>
</head>
<body>

  <!-- Headings demonstration -->
  <h1>Main Chapter Title (Level 1)</h1>
  <h2>Sub-Section Heading (Level 2)</h2>
  <h3>Detailed Topic Heading (Level 3)</h3>

  <!-- Paragraph demonstration -->
  <p>
    This is the first paragraph. HTML documents organize text blocks
    using the paragraph tag, which automatically separates ideas with margins.
  </p>

  <!-- Line break demonstration -->
  <p>
    Postal Address:<br>
    Department of Computer Applications,<br>
    Central College Campus,<br>
    Bengaluru – 560001.
  </p>

</body>
</html>
```

3. Element Characteristics Comparison

Element	Display Type	Closing Tag	Primary Semantic Role	Default Rendering
`<h1> - <h6>`	Block-level	Required (`</h1>`)	Defines page outline and hierarchical document headings.	Bold text with decreasing default font size (2em down to 0.67em).
`<p>`	Block-level	Required (`</p>`)	Groups sentences into distinct readable paragraphs.	Normal weight text with standard top and bottom margins (1em).
` `	Inline (Void)	None (Self-closing ` `)	Forces an immediate text line break within an existing block.	Advances following content to the next line with zero extra margin.

4. Rendered Browser Output

Main Chapter Title (Level 1)

Sub-Section Heading (Level 2)

Detailed Topic Heading (Level 3)

This is the first paragraph. HTML documents organize text blocks using the paragraph tag, which automatically separates ideas with margins.

Postal Address:
Department of Computer Applications,
Central College Campus,
Bengaluru – 560001.

Q13. Explain the font-size, font-style, font-variant, and font-weight properties with examples.

1. Core Concept

CSS typography properties govern the visual styling, proportions, weight, and presentation of rendered typographic glyphs on web pages. The four properties `font-size`, `font-style`, `font-variant`, and `font-weight` provide granular control over text readability and aesthetic emphasis.

2. Detailed Explanation of the Four Properties

- **font-size:** Sets the physical height or dimension of text glyphs. It accepts absolute units (such as `px`, `pt`), relative scalable units (such as `em`, `rem`, `%`, `vw`), and predefined keyword identifiers (such as `small`, `medium`, `large`). Example: `font-size: 18px;`
- **font-style:** Determines whether text characters are rendered upright or slanted. It supports three distinct values: `normal` (standard upright posture), `italic` (specifically designed cursive italic glyphs), and `oblique` (synthetically slanted standard characters). Example: `font-style: italic;`
- **font-variant:** Controls the display of small-caps typographic alternates. The primary values are `normal` (regular lowercase and uppercase glyphs) and `small-caps` (renders lowercase letters as scaled uppercase letters of reduced height). Example: `font-variant: small-caps;`
- **font-weight:** Specifies the boldness or stroke thickness of the typeface. It accepts keyword descriptors (`normal`, `bold`, `bolder`, `lighter`) and standardized numerical values ranging from `100` to `900` in steps of 100, where `400` equates to normal and `700` equates to bold. Example: `font-weight: 700;`

3. Full Working HTML & CSS Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Font Properties Demonstration</title>
  <style>
    .sample-size {
      font-size: 22px;
      color: #1e3a8a;
    }
    .sample-style {
      font-style: italic;
      color: #065f46;
    }
    .sample-variant {
      font-variant: small-caps;
      color: #92400e;
    }
    .sample-weight {
      font-weight: 800;
      color: #7f1d1d;
    }
    .sample-combined {
      font-size: 18px;
      font-style: italic;
      font-variant: small-caps;
      font-weight: bold;
      color: #374151;
    }
  </style>
</head>
<body>
  <p class="sample-size">This text illustrates font-size set to 22px.</p>
  <p class="sample-style">This text illustrates font-style set to italic.</p>
  <p class="sample-variant">This text illustrates font-variant set to small-caps.</p>
  <p class="sample-weight">This text illustrates font-weight set to 800 (Heavy Bold).</p>
  <p class="sample-combined">This text combines all four typographic font properties.</p>
</body>
</html>
```

4. Properties Summary Table

Property Name	Common Allowed Values	Default Value	Visual Effect on Text
`font-size`	`16px`, `1.2rem`, `120%`, `large`	`medium` (16px)	Adjusts the height and proportional width of characters.
`font-style`	`normal`, `italic`, `oblique`	`normal`	Slants characters for emphasis or stylistic differentiation.
`font-variant`	`normal`, `small-caps`	`normal`	Displays lowercase characters as miniature uppercase letters.
`font-weight`	`normal`, `bold`, `100 - 900`	`normal` (400)	Regulates the thickness and boldness of font stroke vectors.

Q14. Explain how to create, access, and modify objects in JavaScript.

1. Core Concept of JavaScript Objects

In JavaScript, an object is a standalone, non-primitive data structure composed of an unordered collection of related properties (key-value pairs) and methods (functions stored as property values). Keys are strings or symbols, and values can be any data type, including primitive numbers, strings, arrays, or nested objects.

2. Object Creation, Access, and Modification Techniques

- **Object Creation:** The most concise and standard technique is using **Object Literal notation** with curly braces `{...}`. Alternatively, objects can be constructed using the `new Object()` constructor or custom constructor functions.
- **Property Access:** Properties are retrieved using two approaches:
  - **Dot Notation:** `object.propertyName` (clean, preferred for valid identifier names).
  - **Bracket Notation:** `object["propertyName"]` (required when keys contain spaces, special characters, or are stored in dynamic variables).
- **Property Modification:** Existing properties are updated via assignment (`object.key = newValue`). New properties are added dynamically at runtime by assigning to a previously non-existent key name.
- **Property Deletion:** Properties are removed from an object using the `delete` operator (`delete object.key`).

3. Full Working JavaScript Code Example

```
// 1. Creating an object using Object Literal notation
const student = {
  rollNo: 101,
  firstName: "Nikhil",
  lastName: "Kumar",
  course: "BCA",
  semester: 5,
  marks: [85, 92, 78],
  // Method definition inside object
  getFullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

// 2. Accessing object properties
console.log("Dot Notation:", student.firstName);         // Output: Nikhil
console.log("Bracket Notation:", student["course"]);      // Output: BCA
console.log("Invoking Method:", student.getFullName());   // Output: Nikhil Kumar

// Dynamic access using variable key
const keyName = "semester";
console.log("Dynamic Key Access:", student[keyName]);     // Output: 5

// 3. Modifying existing properties
student.semester = 6;
student["rollNo"] = 105;

// 4. Adding new properties dynamically
student.email = "nikhil@example.com";
student["gpa"] = 3.9;

// 5. Deleting a property
delete student.lastName;

// Logging the updated object
console.log("Updated Object:", student);
```

4. Operations Summary Table

Action	Syntax Pattern	Code Example	Execution Result
Creation	`const obj = { key: val };`	`const car = { brand: "Tesla", year: 2023 };`	Instantiates object in heap memory.
Dot Access	`obj.key`	`let b = car.brand;`	Returns value `"Tesla"`.
Bracket Access	`obj["key"]`	`let y = car["year"];`	Returns value `2023`.
Modification	`obj.key = newVal;`	`car.year = 2024;`	Updates property `year` to 2024.
Addition	`obj.newKey = val;`	`car.color = "Red";`	Appends `color` property dynamically.
Deletion	`delete obj.key;`	`delete car.brand;`	Removes `brand` property entirely.

Q15. Explain the important mouse events supported by JavaScript.

1. Core Concept and Event Architecture

In client-side JavaScript, mouse events are asynchronous system notifications generated by the browser when a user interacts with DOM elements using a pointing peripheral device (such as a physical mouse or trackpad). Event listener functions are attached to target elements to detect these interactions and execute corresponding program logic.

2. Detailed Breakdown of Major Mouse Events

- **click:** Triggers when the primary pointing device button (usually the left button) is pressed down and released over the exact same element. Commonly used for buttons, links, and form submissions.
- **dblclick:** Triggers when the primary button is clicked twice in rapid succession over the same element. Frequently used for editing inline text or expanding folder items.
- **mousedown:** Triggers the instant any mouse button is pressed down, prior to release. Useful for dragging behaviors and drawing tools.
- **mouseup:** Triggers the instant a pressed mouse button is released over an element. Used in conjunction with `mousedown` to finalize drag-and-drop operations.
- **mouseover:** Triggers when the mouse pointer moves onto an element or into one of its child elements. Supports event bubbling.
- **mouseout:** Triggers when the mouse pointer moves completely out of an element or out of one of its child elements.
- **mousemove:** Triggers continuously whenever the mouse pointer moves by even one pixel while remaining over the element, providing coordinates (`clientX` , `clientY`).

3. Full Working HTML & JavaScript Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Mouse Events Demonstration</title>
  <style>
    .interactive-box {
      width: 320px;
      height: 120px;
      background-color: #e2e8f0;
      border: 2px solid #64748b;
      display: flex;
      align-items: center;
      justify-content: center;
      font-weight: 600;
      color: #1e293b;
      cursor: pointer;
      transition: background-color 0.2s;
    }
  </style>
</head>
<body>

  <div id="eventBox" class="interactive-box">
    Interact with me using mouse
  </div>
  <p id="statusLog">Event Log: None</p>

  <script>
    const box = document.getElementById("eventBox");
    const log = document.getElementById("statusLog");

    // Click event
    box.addEventListener("click", function() {
      log.textContent = "Event Log: Single Click detected!";
      box.style.borderColor = "#2563eb";
    });

    // Double click event
    box.addEventListener("dblclick", function() {
      log.textContent = "Event Log: Double Click detected!";
      box.style.backgroundColor = "#fed7aa";
    });

    // Mouse over event
    box.addEventListener("mouseover", function() {
      box.style.backgroundColor = "#bbf7d0";
      log.textContent = "Event Log: Mouse Entered (mouseover)";
    });

    // Mouse out event
    box.addEventListener("mouseout", function() {
      box.style.backgroundColor = "#e2e8f0";
      log.textContent = "Event Log: Mouse Exited (mouseout)";
    });

    // Mouse down event
    box.addEventListener("mousedown", function() {
      box.style.backgroundColor = "#fef08a";
      log.textContent = "Event Log: Mouse Button Pressed Down (mousedown)";
```

```
    });

    // Mouse up event
    box.addEventListener("mouseup", function() {
      log.textContent = "Event Log: Mouse Button Released (mouseup)";
    });
  </script>

</body>
</html>
```

4. Mouse Events Reference Matrix

Event Name	Inline HTML Attribute	Triggering Condition	Common Application
`click`	`onclick`	Mouse button pressed and released on element.	Button clicks, form submission, modal dialog open.
`dblclick`	`ondblclick`	Two rapid consecutive clicks on element.	Word selection, card zoom, desktop folder open.
`mousedown`	`onmousedown`	Instantaneous mouse button depression.	Canvas drawing start, slider drag initialization.
`mouseup`	`onmouseup`	Instantaneous mouse button release.	Dropping items, finishing drawing stroke.
`mouseover`	`onmouseover`	Cursor enters element boundary.	Drop-down navigation menus, hover cards.
`mouseout`	`onmouseout`	Cursor exits element boundary.	Hiding tooltips, resetting menu state.
`mousemove`	`onmousemove`	Cursor moves within element bounds.	Tracking cursor coordinates, interactive canvas games.

Q16. Explain the different categories of operators in PHP with examples.

1. Core Concept of PHP Operators

Operators in PHP are special programmatic symbols and keywords that instruct the PHP interpreter to perform specific mathematical, logical, comparison, or string operations upon one, two, or three operands to yield a computed value.

2. Major Categories of PHP Operators

- **Arithmetic Operators:** Perform standard mathematical calculations on numerical operands: `+` (Addition), `-` (Subtraction), `*` (Multiplication), `/` (Division), `%` (Modulus remainder), and `**` (Exponentiation).
- **Assignment Operators:** Assign values to variables. Includes basic assignment (`=`) and compound shortcut operators: `+=` , `-=` , `*=` , `/=` , `%=` , and `.=` (concatenation assignment).
- **Comparison Operators:** Compare two values and evaluate to a boolean (`true` or `false`): `==` (equal value), `===` (identical value and data type), `!=` or `<>` (not equal), `!==` (not identical), `<` , `>` , `<=` , `>=` , and `<=>` (spaceship operator returning -1, 0, or 1).
- **Increment / Decrement Operators:** Modify an integer variable by 1: `++$x` (pre-increment), `$x++` (post-increment), `--$x` (pre-decrement), and `$x--` (post-decrement).
- **Logical Operators:** Combine conditional statements: `&&` or `and` (Logical AND), `||` or `or` (Logical OR), `!` (Logical NOT), and `xor` (Exclusive OR: true if either is true, but not both).
- **String Operators:** Specifically designed for string manipulation: Concatenation operator (`.`) which joins two strings, and concatenation assignment operator (`.=`) which appends the right argument to the left argument.

3. Full Working PHP Code Example

```php
<?php
// 1. Arithmetic Operators
$a = 15;
$b = 4;
echo "Addition: " . ($a + $b) . "\n";        // 19
echo "Modulus: " . ($a % $b) . "\n";         // 3
echo "Exponentiation: " . ($a ** 2) . "\n";  // 225

// 2. Assignment Operators
$x = 20;
$x += 10; // Equivalent to $x = $x + 10;
echo "Compound Assignment: " . $x . "\n";    // 30

// 3. Comparison Operators (Loose vs Strict)
$num1 = 100;
$num2 = "100";
var_dump($num1 == $num2);   // bool(true) - loose equality (same value)
var_dump($num1 === $num2);  // bool(false) - strict equality (different types: int vs string)

// Spaceship operator
echo (5 <=> 10) . "\n";     // -1 (left is smaller)
echo (10 <=> 10) . "\n";    // 0 (both are equal)
echo (15 <=> 10) . "\n";    // 1 (left is greater)

// 4. Increment / Decrement Operators
$val = 5;
echo "Post-increment: " . $val++ . "\n";     // Outputs 5, then increments to 6
echo "After post-increment: " . $val . "\n"; // Outputs 6
echo "Pre-increment: " . ++$val . "\n";      // Increments to 7, then outputs 7

// 5. Logical Operators
$age = 22;
$hasVoterID = true;
if ($age >= 18 && $hasVoterID) {
    echo "Eligible to vote in election.\n";
}

// 6. String Operators
$greeting = "Hello";
$greeting .= " World!";     // Appends to existing string
echo $greeting . "\n";      // Outputs: Hello World!
?>
```

4. Operators Categorical Matrix

Operator Category	Symbols Included	Syntax Example	Evaluated Output
Arithmetic	`+`, `-`, `*`, `/`, `%`, `**`	`$res = 10 % 3;`	Returns remainder `1`.
Assignment	`=`, `+=`, `-=`, `*=`, `.=`	`$x += 5;`	Adds 5 to `$x` and stores in `$x`.
Comparison	`==`, `===`, `!=`, `<>`, `<=>`	`$res = (5 === "5");`	Returns boolean `false`.
Increment/Decrement	`++$x`, `$x++`, `--$x`, `$x--`	`$res = ++$val;`	Increments `$val` before returning.
Logical	`&&`, `||`, `!`, `xor`	`$res = ($p && $q);`	Returns `true` only if both operands are true.
String	`.`, `.=`	`$s = "Web " . "App";`	Concatenates into `"Web App"`.

MODEL QUESTION PAPER – 5

> **Q11. Explain hyperlinks using the anchor element. Describe external links, internal links, email links, and image hyperlinks with examples.**

1. Core Concept and the Anchor Element

Hyperlinks connect web resources across the Internet, enabling users to jump between pages, navigate within a document, or trigger network protocols. In HTML, hyperlinks are defined using the `<a>` (anchor) element.

- **href:** Specifies the destination URL, file path, or section identifier.
- **target:** Specifies the browsing context for opening the document (e.g., `_blank` opens in a new tab; `_self` opens in the same frame).
- **title:** Provides supplementary advisory information displayed as a tooltip on hover.

2. Types of Hyperlinks

- **External Links:** Point to external websites or documents hosted on different domains. They require absolute URLs including the transfer protocol (e.g., `https://`).
- **Internal Links:** Point to pages within the same website using relative paths, or jump to specific bookmark locations within the same page using element IDs prefixed with a hash (`#id`).
- **Email Links:** Utilize the `mailto:` URL scheme to open the client's default email client pre-addressed to a specified email address.
- **Image Hyperlinks:** Created by nesting an `<img>` element inside an `<a>` tag, rendering the image as a clickable navigation control.

3. Complete Working HTML Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Demonstration of HTML Hyperlinks</title>
</head>
<body>
  <h1 id="page-top">HTML Hyperlink Demonstrations</h1>

  <!-- 1. External Hyperlink -->
  <p>
    <strong>External Link:</strong>
    <a href="https://www.w3.org" target="_blank" rel="noopener noreferrer" title="W3C Official Site">Visit W3C
  </p>

  <!-- 2. Internal Page Bookmark Link -->
  <p>
    <strong>Internal Jump Link:</strong>
    <a href="#footer-section">Jump to Footer Contact Info</a>
  </p>

  <!-- 3. Email Hyperlink -->
  <p>
    <strong>Email Link:</strong>
    <a href="mailto:principal@college.edu?subject=Admission%20Inquiry">Send Admission Inquiry Email</a>
  </p>

  <!-- 4. Image Hyperlink -->
  <p>
    <strong>Image Hyperlink:</strong><br>
    <a href="https://www.google.com" target="_blank">
      <img src="search_icon.png" alt="Google Search Portal" width="120" height="40">
    </a>
  </p>

  <div style="margin-top: 400px;" id="footer-section">
    <hr>
    <h3>Footer Section</h3>
    <p>College Contact: contact@college.edu</p>
    <a href="#page-top">Back to Top</a>
  </div>
</body>
</html>
```

4. Browser Output Description

Text links render in blue with an underline by default. Visited links display in purple, and active clicked links render in red. Clicking the image displays a focus outline and routes the browser to Google Search. Clicking the internal link smoothly scrolls the browser viewport to the footer section.

Q12. Explain CSS Grid and its important properties with a suitable example.

1. Core Concept

CSS Grid Layout is a two-dimensional layout system designed for web pages, allowing developers to organize content into structured rows and columns simultaneously. Unlike Flexbox (which is one-dimensional: row or column), CSS Grid manages both axes concurrently.

A grid structure consists of a **Grid Container** (the parent element with `display: grid;`) and **Grid Items** (the direct child elements placed into grid tracks).

2. Important CSS Grid Properties

Property	Applies To	Description & Common Values
`grid-template-columns`	Container	Defines the number and width of column tracks (e.g., `repeat(3, 1fr)`, `200px 1fr`).
`grid-template-rows`	Container	Defines the height of individual row tracks (e.g., `100px auto 60px`).
`gap` / `grid-gap`	Container	Specifies the gutter spacing between grid cells (e.g., `gap: 15px;`).
`grid-column`	Item	Shorthand for `grid-column-start` and `grid-column-end` (e.g., `1 / 3` or `span 2`).
`grid-row`	Item	Shorthand for `grid-row-start` and `grid-row-end` (e.g., `span 2`).

3. Complete Working Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Grid Layout Example</title>
  <style>
    .grid-container {
      display: grid;
      grid-template-columns: 200px 1fr 1fr;
      grid-template-rows: 80px 250px 60px;
      gap: 12px;
      max-width: 800px;
      margin: 20px auto;
      font-family: sans-serif;
      text-align: center;
    }
    .grid-item {
      background-color: #e2e8f0;
      border: 1px solid #94a3b8;
      padding: 15px;
      font-weight: 600;
    }
    .header {
      grid-column: 1 / 4;
      background-color: #38bdf8;
    }
    .sidebar {
      grid-row: 2 / 3;
      background-color: #fcd34d;
    }
    .main-content {
      grid-column: 2 / 4;
      background-color: #86efac;
    }
    .footer {
      grid-column: 1 / 4;
      background-color: #cbd5e1;
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div class="grid-item header">Header (Spans 3 Columns)</div>
    <div class="grid-item sidebar">Sidebar (Row 2, Col 1)</div>
    <div class="grid-item main-content">Main Content Area (Spans Columns 2 and 3)</div>
    <div class="grid-item footer">Footer (Spans 3 Columns)</div>
  </div>
</body>
</html>
```

4. Layout Output Structure

The container creates a rigid 3-column layout. The header spans all 3 columns across row 1. Row 2 holds the 200px sidebar in column 1, while the main content expands across columns 2 and 3 using fractional units. Row 3 contains the full-width footer spanning columns 1 to 4.

Q13. Explain the top, right, bottom, left, and z-index properties.

1. Core Concept of Positioning Properties

The CSS offset properties (`top` , `right` , `bottom` , and `left`) specify the physical offset distance of a positioned element from the edges of its reference box.
Essential Prerequisite: These four properties have **no effect** on elements with `position: static;` (the default). They only function when `position` is set to `relative` , `absolute` , `fixed` , or `sticky` .

2. Offset Properties Breakdown

- **top:** Moves the element downward relative to the top reference edge (e.g., `top: 20px;`).
- **right:** Moves the element leftward relative to the right reference edge (e.g., `right: 15px;`).
- **bottom:** Moves the element upward relative to the bottom reference edge (e.g., `bottom: 0px;`).
- **left:** Moves the element rightward relative to the left reference edge (e.g., `left: 50px;`).

Reference frame depends on positioning mode:

- `relative:` Shifted relative to its own natural position in the document flow.
- `absolute:` Shifted relative to its closest positioned ancestor (non-static).
- `fixed:` Shifted relative to the browser viewport window.

3. The z-index Property (Stacking Order)

The `z-index` property specifies the vertical stacking order of overlapping positioned elements along the z-axis (perpendicular to the screen).

- **Default Value:** `auto` (rendered in natural HTML source order).
- **Syntax:** `z-index: <integer>;` (accepts positive, zero, or negative numbers).
- **Rule:** An element with a higher numeric `z-index` renders in front of an element with a lower `z-index` when elements occupy overlapping screen space.

4. Complete Working Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Positioning and z-index Example</title>
  <style>
    .container {
      position: relative;
      width: 400px;
      height: 250px;
      border: 2px dashed #475569;
      margin: 30px auto;
      background-color: #f8fafc;
    }
    .box {
      position: absolute;
      width: 140px;
      height: 100px;
      padding: 10px;
      font-family: sans-serif;
      font-size: 13px;
      font-weight: bold;
      color: #ffffff;
      border-radius: 4px;
    }
    .box1 {
      top: 20px;
      left: 20px;
      background-color: #ef4444; /* Red */
      z-index: 1;
    }
    .box2 {
      top: 60px;
      left: 70px;
      background-color: #3b82f6; /* Blue */
      z-index: 3; /* Highest z-index: renders on top */
    }
    .box3 {
      top: 100px;
      left: 120px;
      background-color: #10b981; /* Green */
      z-index: 2; /* Sits between Box 1 and Box 2 */
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="box box1">Box 1<br>z-index: 1</div>
    <div class="box box2">Box 2<br>z-index: 3 (Top)</div>
    <div class="box box3">Box 3<br>z-index: 2</div>
  </div>
</body>
</html>
```

5. Stacking Order Output

Although Box 3 appears last in HTML source order, Box 2 renders visibly on top of Box 3 and Box 1 because its `z-index: 3` is highest. Box 3 sits in the middle layer (`z-index: 2`), and Box 1 rests at the bottom of the stack (`z-index: 1`).

Q14. Write a JavaScript program to generate the factorial of a number using loops.

1. Mathematical Logic

The factorial of a non-negative integer `n` (written as `n!`) is the product of all positive integers less than or equal to `n`:
`n! = n × (n - 1) × (n - 2) × ... × 1`
Special boundary conditions:

- `0! = 1` and `1! = 1`.
- Factorials are undefined for negative numbers.

2. Algorithm Using a Loop

- **Step 1:** Read the input integer `n` from user input.
- **Step 2:** Check if `n < 0`; if true, display an error message.
- **Step 3:** Initialize variable `fact = 1`.
- **Step 4:** Execute a `for` loop from `i = 1` up to `n`, calculating `fact = fact * i`.
- **Step 5:** Return and print the calculated factorial value.

3. Complete Working HTML and JavaScript Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Factorial Calculator</title>
  <style>
    body { font-family: Arial, sans-serif; padding: 25px; }
    .card { max-width: 420px; padding: 20px; border: 1px solid #cbd5e1; border-radius: 6px; }
    input[type="number"] { width: 100%; padding: 8px; margin: 8px 0; box-sizing: border-box; }
    button { background-color: #2563eb; color: #fff; padding: 10px 16px; border: none; border-radius: 4px; cur
    button:hover { background-color: #1d4ed8; }
    #result { margin-top: 15px; font-weight: bold; font-size: 15px; color: #0f172a; }
  </style>
</head>
<body>

  <div class="card">
    <h3>Factorial Generator</h3>
    <label for="numInput">Enter a non-negative integer:</label>
    <input type="number" id="numInput" placeholder="e.g. 5" min="0">
    <button onclick="computeFactorial()">Calculate Factorial</button>
    <div id="result"></div>
  </div>

  <script>
    function computeFactorial() {
      var n = parseInt(document.getElementById("numInput").value);
      var resultDiv = document.getElementById("result");

      // Validation
      if (isNaN(n)) {
        resultDiv.innerHTML = "<span style='color:red;'>Please enter a valid number.</span>";
        return;
      }
      if (n < 0) {
        resultDiv.innerHTML = "<span style='color:red;'>Factorial is not defined for negative numbers.</span>"
        return;
      }

      // Loop Computation
      var fact = 1;
      for (var i = 1; i <= n; i++) {
        fact = fact * i;
      }

      resultDiv.innerHTML = n + "! = " + fact;
    }
  </script>

</body>
</html>
```

4. Sample Output and Dry Run

- **Input:** `5` → Loop steps: `1 × 1 = 1` → `1 × 2 = 2` → `2 × 3 = 6` → `6 × 4 = 24` → `24 × 5 = 120`.
- **Rendered Output:** `5! = 120`
- **Input:** `0` → Loop does not execute → Output: `0! = 1`

Q15. Explain how JavaScript works in a web browser.

1. Core Concept and Execution Environment

JavaScript is a single-threaded, synchronous, interpreted or JIT-compiled client-side scripting language. When a web browser loads a webpage, the browser engine and its embedded JavaScript Engine (such as Google Chrome's V8 or Mozilla's SpiderMonkey) collaborate to parse HTML, execute JavaScript, manipulate the Document Object Model (DOM), and handle asynchronous user events.

2. Architecture of the Browser Runtime Environment

The browser runtime comprises several coordinated components:

- **JavaScript Engine:** Contains the **Call Stack** (which tracks function execution frames in a LIFO order) and the **Memory Heap** (which allocates memory dynamically for objects and variables).
- **Web APIs:** Browser-provided features outside the core JS engine (DOM API, Fetch / XMLHttpRequest, `setTimeout()`, Geolocation) running on background browser threads.
- **Callback Queue (Task Queue):** Stores completed asynchronous callbacks ready for execution.
- **Event Loop:** A continuous monitoring process that checks if the Call Stack is empty; if empty, it pushes the first waiting task from the Callback Queue onto the Call Stack.

3. Step-by-Step Execution Lifecycle

```
[HTML Parser] ---> Encounter <script> tag ---> Download JS File
                                                        |
[AST Generation] <--- Tokenize & Parse Code <------------+
       |
[JIT Compilation] ---> Bytecode / Machine Code Generation
                                |
[Call Stack Execution] <--------+
       |
       +---> Synchronous Calls execute immediately (LIFO)
       +---> Asynchronous Calls delegated to Web APIs (Timers, AJAX)
                    |
             [Web API finishes] ---> Pushes callback to [Callback Queue]
                                                              |
             [Event Loop] transfers callback to [Call Stack] when empty
```

4. Illustrative Code Example

```
console.log("Step 1: Start");

setTimeout(function() {
  console.log("Step 2: Timeout Callback");
}, 2000);

console.log("Step 3: End");

// Browser Output:
// Step 1: Start
// Step 3: End
// (after 2 seconds delay)
// Step 2: Timeout Callback
```

In this example, `console.log("Step 1")` executes immediately in the Call Stack. `setTimeout` registers its timer with the browser's Web API and pops off the stack. `console.log("Step 3")` executes synchronously. After 2000ms, the Web API moves the callback to the Task Queue, where the Event Loop pushes it onto the Call Stack for execution.

Q16. Explain the methods of copying, moving, deleting, and updating files in PHP.

1. Core Concept of File Handling in PHP

PHP executes on the web server and provides built-in filesystem functions to manipulate files and directories directly on the host server. To perform file modifications successfully, the web server user process (e.g., Apache/Nginx) requires appropriate read, write, and execute permissions on the target directory path.

2. File Operation Methods in PHP

- **Copying Files (`copy()`):** Duplicates a file from a source path to a destination path.
  `bool copy(string $source, string $dest);` — Returns `true` on success and `false` on failure.
- **Moving / Renaming Files (`rename()`):** Moves a file to another directory or renames it in place.
  `bool rename(string $oldname, string $newname);` — Returns `true` on success.
- **Deleting Files (`unlink()`):** Permanently removes a file from the server filesystem.
  `bool unlink(string $filename);` — Returns `true` on success.
- **Updating Files:** Updating involves either overwriting existing contents or appending new data. Common approaches:
  - `fopen()` with mode `'w'` (overwrite) or `'a'` (append), followed by `fwrite()` and `fclose()`.
  - `file_put_contents($file, $data, FILE_APPEND);` — A convenient helper function that writes data in one step.

3. Complete PHP Demonstration Script

```php
<?php
// Target file names
$originalFile = "data.txt";
$copiedFile   = "backup_data.txt";
$movedFile    = "archive/data_archived.txt";

// 1. CREATE / INITIALIZE FILE
file_put_contents($originalFile, "Initial Record: BCA 5th Sem\n");
echo "1. File created successfully.
";

// 2. UPDATE FILE (Appending Data)
$newData = "Updated Record: Web Programming DCCA511\n";
file_put_contents($originalFile, $newData, FILE_APPEND);
echo "2. File updated (appended) successfully.
";

// 3. COPY FILE
if (copy($originalFile, $copiedFile)) {
    echo "3. File successfully copied to " . $copiedFile . ".
";
} else {
    echo "Failed to copy file.
";
}

// 4. MOVE / RENAME FILE
// Ensure destination directory exists
if (!is_dir("archive")) {
    mkdir("archive");
}
if (rename($copiedFile, $movedFile)) {
    echo "4. File successfully moved to " . $movedFile . ".
";
} else {
    echo "Failed to move file.
";
}

// 5. DELETE FILE
if (file_exists($originalFile)) {
    if (unlink($originalFile)) {
        echo "5. Original file " . $originalFile . " successfully deleted.
";
    }
}
?>
```

4. Summary of Functions

Operation	PHP Function	Return Type	Primary Purpose
Copy	`copy($src, $dst)`	`bool`	Creates an exact clone of the file at destination.
Move / Rename	`rename($old, $new)`	`bool`	Moves file across directories or changes file name.
Delete	`unlink($file)`	`bool`	Permanently deletes file from disk.
Update	`file_put_contents()` / `fwrite()`	`int|bool`	Appends or replaces text data inside target file.

PART III: SECTION C — 8 MARKS QUESTIONS (PAPERS 1 TO 5)

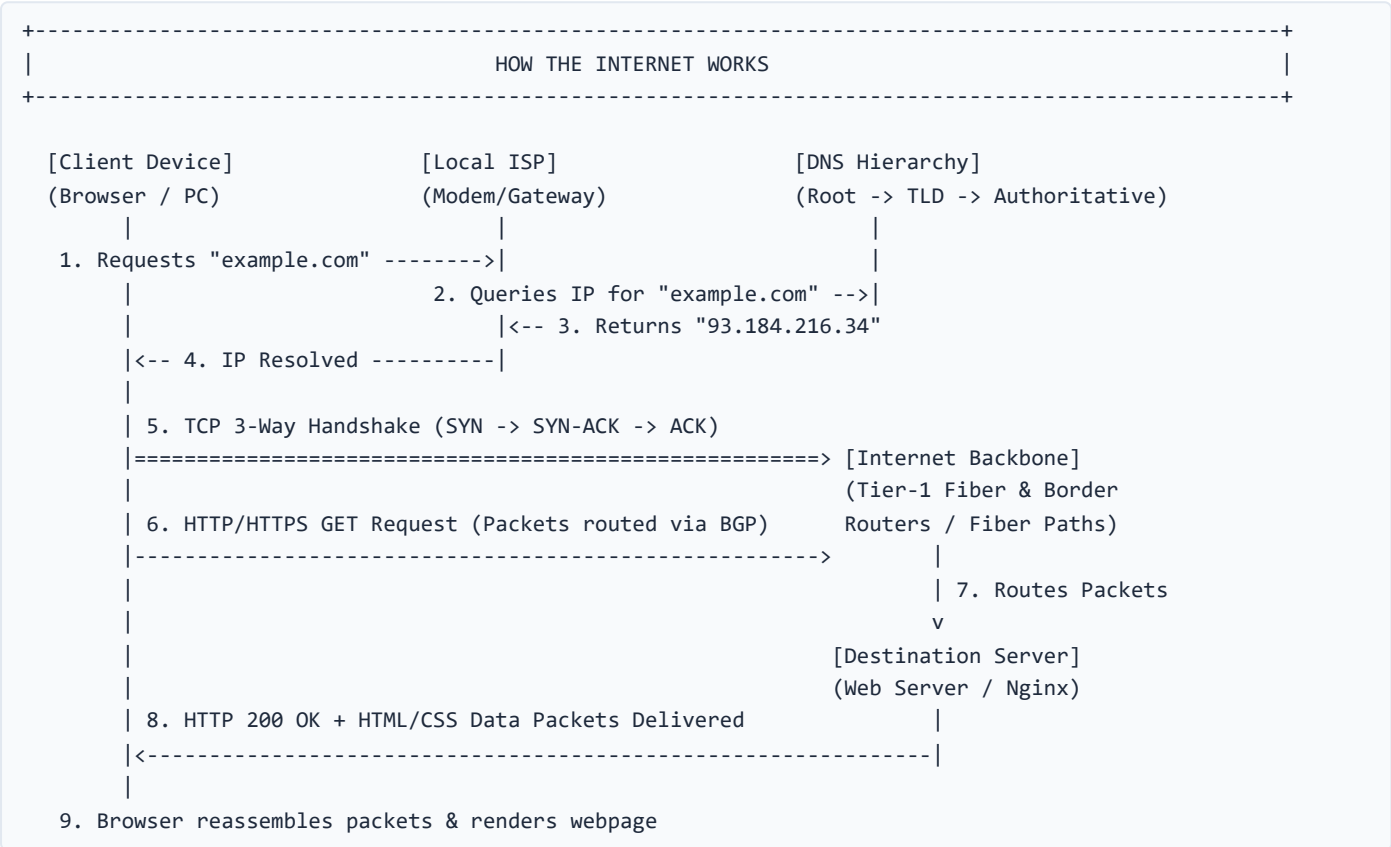
MODEL QUESTION PAPER – 1

Q17. Explain how the Internet works with a suitable diagram.

1. Core Concept

The Internet is a globally distributed, decentralized network of interconnected computers and autonomous networks. It operates on the packet-switching principle governed by the standardized Internet Protocol Suite (TCP/IP). When a client requests web resources, communications traverse a structured sequence: Client → Local Internet Service Provider (ISP) → Domain Name System (DNS) → Tier-1 Backbone Routers → Destination Web Server.

2. Architecture Workflow Diagram



3. Detailed Step-by-Step Working Mechanism

1. **User Request & URL Parsing:** The user enters a Uniform Resource Locator (e.g., `https://www.example.com`) into the browser. The browser breaks the URL into protocol (HTTPS), domain name (`example.com`), and requested path.
2. **DNS Resolution:** Computers communicate via numerical IP addresses. The client queries its local DNS cache; if not found, it queries the ISP recursive resolver. The resolver contacts Root DNS servers, Top-Level Domain (TLD) servers (`.com`), and Authoritative Name Servers to translate the alphanumeric hostname into an IP address (e.g., `93.184.216.34`).
3. **TCP Connection & 3-Way Handshake:** To establish a dependable communication channel, the client initiates a TCP 3-way handshake with the server's IP:
   - **SYN:** Client sends a synchronization flag packet.
   - **SYN-ACK:** Server acknowledges and sends synchronization flag.
   - **ACK:** Client acknowledges receipt. A secure TLS/SSL handshake follows for HTTPS.

4. **Packet Switching & Routing:** The client's HTTP request is split into small units called packets. Each packet contains source/destination IP headers, sequence identifiers, and payload. Packets navigate across international Internet Exchange Points (IXPs) and routers using protocols like BGP (Border Gateway Protocol) and OSPF, dynamically avoiding network congestion.
5. **Server Processing:** The destination server processes the request, executes back-end logic (e.g., PHP scripts, database queries), and creates an HTTP 200 OK response with the requested payload.
6. **Packet Reassembly & Page Rendering:** TCP on the client machine inspects incoming packets, verifies checksums, requests retransmission of dropped packets, and reassembles them sequentially. The browser parses the HTML, executes scripts, and renders the visual webpage.

4. Core Protocol Stack Summary

Protocol Layer	Representative Protocols	Primary Function
Application Layer	HTTP, HTTPS, DNS, FTP, SMTP	Direct interface with user applications and web browsers.
Transport Layer	TCP, UDP	Host-to-host flow control, reliability, error checking, packet ordering.
Internet Layer	IP (IPv4, IPv6), ICMP	Logical packet addressing, path determination, and internetwork routing.
Network Access Layer	Ethernet, Wi-Fi, Fiber Optic cables	Physical transmission of binary frames across physical communication media.

Q18. Explain the basic structure of an HTML document with suitable code.

1. Core Concept

An HTML document is a structured tree of nested elements conforming to the W3C HTML5 standard. It clearly bifurcates document configuration metadata (contained in `<head>`) from human-viewable content and structural layout (contained in `<body>`).

2. Semantic Document Architecture

```
+-----------------------------------------------------------------------+
| <!DOCTYPE html>                                                       |
| <html lang="en">                                                      |
|   <head>                                                              |
|     <meta charset="UTF-8">                                            |
|     <title>Page Title in Browser Tab</title>                          |
|   </head>                                                             |
|   <body>                                                              |
|     <header> Header / Branding Banner </header>                       |
|     <nav> Navigation Menus </nav>                                     |
|     <main>                                                            |
|       <section> Primary Document Content </section>                   |
|       <aside> Sidebar / Related Links </aside>                        |
|     </main>                                                           |
|     <footer> Copyright & Contact Information </footer>                |
|   </body>                                                             |
| </html>                                                               |
+-----------------------------------------------------------------------+
```

3. Detailed Explanation of Structural Elements

- **<!DOCTYPE html>:** The document type declaration placed at the top of the file. It informs modern web browsers to render the page in standard HTML5 mode rather than legacy quirks mode.
- **<html lang="en">:** The root element enclosing all content. The `lang="en"` attribute declares English as the document language for screen readers and search engines.

- **<head>:** The container for document metadata that is not visible on the webpage canvas, except for the page title. Essential head tags include:
  - `<meta charset="UTF-8">`: Sets universal character encoding covering almost all written alphabets.
  - `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Ensures responsive display scaling across mobile and desktop viewports.
  - `<title>`: Specifies the title displayed on browser tabs, bookmarks, and search engine snippets.
  - `<link>`: Attaches external CSS stylesheets; `<script>` loads JavaScript logic.
- **<body>:** Encloses all visual and interactive elements rendered on screen. Modern HTML5 structures body elements using semantic tags:
  - `<header>`: Introductory content, mastheads, or navigation headers.
  - `<nav>`: Section containing navigational hyperlinks.
  - `<main>`: Unique primary content of the document.
  - `<section>` and `<article>`: Thematic sections and self-contained content units.
  - `<footer>`: Concluding information, author details, copyright, and legal links.

4. Complete Working Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Information Portal</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; line-height: 1.6; }
    header { background-color: #0f172a; color: white; padding: 15px; text-align: center; }
    nav a { margin-right: 15px; color: #2563eb; text-decoration: none; font-weight: bold; }
    main { padding: 20px 0; }
    footer { border-top: 1px solid #ccc; padding-top: 10px; font-size: 13px; color: #666; }
  </style>
</head>
<body>
  <header>
    <h1>Department of Computer Applications</h1>
  </header>

  <nav>
    <a href="#home">Home</a>
    <a href="#courses">Courses</a>
    <a href="#contact">Contact</a>
  </nav>

  <main>
    <section id="home">
      <h2>Welcome to BCA 5th Semester</h2>
      <p>This portal provides comprehensive study materials and lab programs for Web Programming.</p>
    </section>
  </main>

  <footer>
    <p>&copy; 2026 Department of Computer Applications. All Rights Reserved.</p>
  </footer>
</body>
</html>
```

5. Summary Table

Structural Tag	Parent Tag	Rendered to User?	Primary Purpose
`<!DOCTYPE>`	None (Top line)	No	Specifies HTML5 document mode to browser engine.
`<html>`	Document root	No	Encloses the entire HTML document hierarchy.
`<head>`	`<html>`	No (except title)	Maintains page metadata, scripts, styles, and character set.
`<body>`	`<html>`	Yes	Contains all viewable text, images, tables, and media.

Q19. Explain the CSS Box Model with a neat diagram.

1. Core Concept

In CSS, every renderable HTML element is represented as a rectangular box. The **CSS Box Model** defines the modular coordinate system and geometric boundary layers that dictate how an element's dimensions, internal spacing, borders, and external clearances are computed on screen.

2. Box Model Diagram



3. The Four Concentric Layers

- **1. Content Area:** The innermost rectangular core where actual text, images, or child elements reside. Its dimensions are determined by the `width` and `height` properties.
- **2. Padding:** The transparent internal buffer region between the content and the border. Padding inherits the element's background color or image. It can be specified uniformly via `padding: 15px;` or per side: `padding-top`, `padding-right`, `padding-bottom`, `padding-left`.
- **3. Border:** The solid, dashed, or stylized boundary line wrapping directly around the padding. Styled via `border-width`, `border-style`, and `border-color`.
- **4. Margin:** The outermost transparent buffer that separates the element's border from adjacent elements on the webpage. Margins do not take the element's background color. Negative margins pull elements closer, while `margin: 0 auto;` horizontally centers block elements.

4. Box Sizing Calculations: content-box vs border-box

- **Standard Box Model (`box-sizing: content-box` - Default):**
  The specified `width` applies strictly to the content area alone. Any padding or borders added expand the rendered size outward:
  `Total Width = width + padding-left + padding-right + border-left + border-right + margin-left + margin-right`
- **Alternative Box Model (`box-sizing: border-box` - Modern Standard):**
  The specified `width` includes the Content, Padding, and Border. Adding padding or borders compresses the content area inward without expanding the total outer box size.

5. Working Code Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    /* Box 1: Default Content-Box */
    .box-content {
      box-sizing: content-box;
      width: 300px;
      padding: 20px;
      border: 5px solid #2563eb;
      margin: 15px;
      background-color: #dbeafe;
      /* Rendered total width = 300 + 40 + 10 = 350px (excluding margin) */
    }

    /* Box 2: Modern Border-Box */
    .box-border {
      box-sizing: border-box;
      width: 300px;
      padding: 20px;
      border: 5px solid #16a34a;
      margin: 15px;
      background-color: #dcfce7;
      /* Rendered total width = strictly 300px (content area shrinks to 250px) */
    }
  </style>
</head>
<body>
  <div class="box-content">content-box: Rendered width = 350px</div>
  <div class="box-border">border-box: Rendered width = strictly 300px</div>
</body>
</html>
```

6. Box Model Layer Comparison

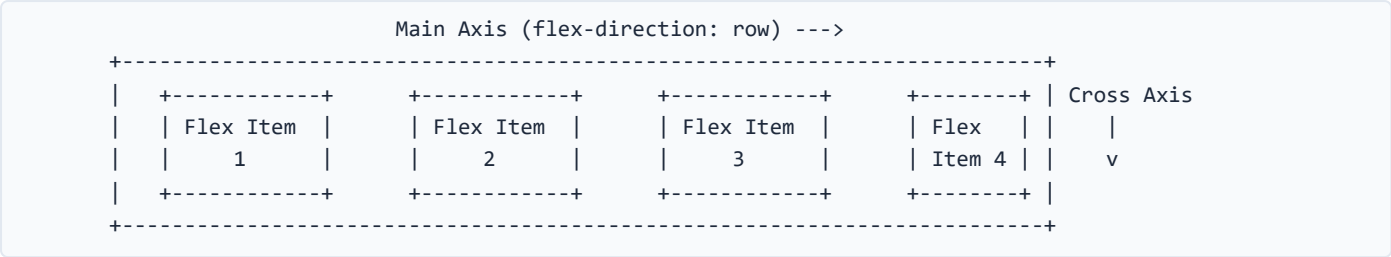
Layer	Location	Background Visible?	Clickable Area?
Content	Innermost core	Yes	Yes
Padding	Between content & border	Yes	Yes
Border	Between padding & margin	Border color/style	Yes
Margin	Outermost buffer	No (transparent)	No

Q20. Explain Flexbox and its important properties with a suitable example.

1. Core Concept

The **Flexible Box Layout (Flexbox)** is a CSS3 one-dimensional layout model designed to distribute space dynamically and align items efficiently along a single axis (row or column). It resolves alignment and sizing challenges in responsive design without relying on floats or positioning hacks.

2. Flexbox Axes & Architecture



3. Flex Container Properties

- **display: flex;** Initializes the flex formatting context for all direct children.
- **flex-direction:** Establishes the main axis direction. Values: `row` (horizontal left-to-right [default]), `row-reverse`, `column` (vertical top-to-bottom), `column-reverse`.
- **justify-content:** Aligns flex items along the **Main Axis**. Values:
  - `flex-start`: Items packed toward the start.
  - `flex-end`: Items packed toward the end.
  - `center`: Items centered along the axis.
  - `space-between`: First item at start, last at end, equal spacing between.
  - `space-around`: Equal space around every item.
  - `space-evenly`: Equal space between any two items and edges.
- **align-items:** Aligns flex items along the perpendicular **Cross Axis**. Values: `stretch` (default), `flex-start`, `flex-end`, `center`, `baseline`.
- **flex-wrap:** Controls whether items break onto multiple lines when container space is exhausted. Values: `nowrap` (default), `wrap`, `wrap-reverse`.
- **gap:** Defines the explicit gutter spacing between items (e.g., `gap: 15px;`).

4. Flex Item Properties

- **flex-grow:** Proportional factor dictating how much an item will expand relative to sibling items when positive free space exists (e.g., `flex-grow: 1;`).
- **flex-shrink:** Dictates the willingness of an item to shrink when container space is insufficient (default: `1`).
- **flex-basis:** Sets the initial default size of an item before space distribution (e.g., `flex-basis: 200px;`).
- **flex (Shorthand):** Combines grow, shrink, and basis: `flex: 1 1 200px;`.
- **align-self:** Overrides the container's `align-items` for an individual flex item.

5. Working Code Example (Responsive Navigation & Card Layout)

```
<!DOCTYPE html>
<html>
<head>
  <style>
    /* Flex Container: Navigation Bar */
    .navbar {
      display: flex;
      justify-content: space-between;
      align-items: center;
      background-color: #0f172a;
      padding: 12px 20px;
      color: white;
    }
    .nav-links {
      display: flex;
      gap: 20px;
      list-style: none;
      margin: 0;
      padding: 0;
    }

    /* Flex Container: Cards Row */
    .card-container {
      display: flex;
      flex-wrap: wrap;
      justify-content: space-between;
      gap: 15px;
      padding: 20px 0;
    }
    .card {
      flex: 1 1 calc(33.333% - 15px);
      background-color: #f1f5f9;
      border: 1px solid #cbd5e1;
      padding: 15px;
      border-radius: 6px;
      min-width: 200px;
    }
  </style>
</head>
<body>
  <div class="navbar">
    <div><strong>Portal Logo</strong></div>
    <ul class="nav-links">
      <li>Home</li>
      <li>Courses</li>
      <li>Projects</li>
      <li>Contact</li>
    </ul>
  </div>

  <div class="card-container">
    <div class="card">HTML5 Fundamentals</div>
    <div class="card">CSS3 Flexbox & Grid</div>
    <div class="card">JavaScript DOM Scripting</div>
  </div>
</body>
</html>
```

> **Q21. Explain how a function is defined in JavaScript. Discuss function calls, passing parameters, and return values.**

1. Core Concept

A function in JavaScript is an executable, modular block of code designed to perform a specific task. Functions prevent redundant coding (DRY principle), support modular encapsulation, and run when invoked (called).

2. Methods of Defining Functions in JavaScript

- **1. Function Declaration (Named Function):**
  Declared with the `function` keyword. Function declarations are hoisted to the top of their scope, allowing them to be called before they appear in source code.
  `function calculateArea(length, breadth) { return length * breadth; }`
- **2. Function Expression (Anonymous Function):**
  Defined as an expression and assigned to a variable. They are not hoisted; they can only be invoked after assignment.
  `const calculateArea = function(length, breadth) { return length * breadth; };`
- **3. Arrow Function (ES6):**
  Provides a compact syntax using the `=>` token. Single-line expressions return implicitly without requiring the `return` keyword or curly braces.
  `const calculateArea = (length, breadth) => length * breadth;`

3. Function Calls, Parameters, and Return Values

- **Function Call (Invocation):** Executed by suffixing the function identifier with parentheses containing arguments: `calculateArea(10, 5);`.
- **Parameters vs Arguments:** Parameters are placeholder identifiers declared in the function definition. Arguments are concrete values passed during invocation.
- **Default Parameters (ES6):** Assigns fallback values if arguments are omitted: `function greet(name = "Student") { ... }`.
- **Rest Parameters (ES6):** Collects variable numbers of arguments into an array using `...`: `function sumAll(...numbers) { ... }`.
- **Return Values:** The `return` statement immediately stops function execution and returns a result to the calling environment. Functions without an explicit return statement return `undefined`.

4. Comprehensive Working JavaScript Program

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Functions Demonstration</title>
</head>
<body>
  <h2>JavaScript Functions Output</h2>
  <div id="output"></div>

  <script>
    // 1. Function Declaration with Return Value
    function add(a, b) {
      return a + b;
    }

    // 2. Function Expression with Default Parameters
    const calculateTotal = function(price, taxRate = 0.18) {
      return price + (price * taxRate);
    };

    // 3. Arrow Function with Implicit Return
    const square = n => n * n;

    // 4. Function using Rest Parameters (...)
    function sumAll(...values) {
      let total = 0;
      for (let val of values) {
        total += val;
      }
      return total;
    }

    // Function Invocations (Calls)
    let sumResult   = add(15, 25);
    let billTotal   = calculateTotal(1000); // Uses default 0.18 tax
    let sqResult    = square(9);
    let totalSum    = sumAll(10, 20, 30, 40, 50);

    // Present Results
    let html = "<ul>" +
      "<li><strong>add(15, 25):</strong> " + sumResult + "</li>" +
      "<li><strong>calculateTotal(1000):</strong> " + billTotal + "</li>" +
      "<li><strong>square(9):</strong> " + sqResult + "</li>" +
      "<li><strong>sumAll(10, 20, 30, 40, 50):</strong> " + totalSum + "</li>" +
      "</ul>";

    document.getElementById("output").innerHTML = html;
  </script>
</body>
</html>
```

5. Function Types Comparison

Category	Syntax Pattern	Hoisted?	this Context
Declaration	`function name(params) { ... }`	Yes	Dynamic (depends on caller)
Expression	`const name = function(params) { ... };`	No	Dynamic (depends on caller)
Arrow Function	`const name = (params) => { ... };`	No	Lexical (inherited from outer scope)

Q22. Explain form handling in PHP and describe its basic working process.

1. Core Concept

Form handling in PHP is the mechanism by which user inputs collected via HTML form controls (text boxes, radio buttons, dropdowns) are submitted to the web server, extracted via PHP superglobals, sanitized for security, and processed (stored in a database or displayed).

2. Basic Working Process (Lifecycle)

1. **Form Presentation:** The client browser renders an HTML form containing input fields and a submit trigger. The `<form>` element defines the destination endpoint in `action` and transmission protocol in `method`.
2. **Form Submission:** When the user submits, the browser encodes the form inputs into key-value pairs (using the HTML `name` attributes as keys) and sends an HTTP request.
3. **Superglobal Ingestion:** The web server executes the target PHP script. PHP automatically parses the payload into associative superglobal arrays:
   - `$_GET`: Ingests parameters appended to the URL query string.
   - `$_POST`: Ingests data packaged in the HTTP request body.
   - `$_REQUEST`: Merges contents of `$_GET`, `$_POST`, and `$_COOKIE`.
4. **Validation and Sanitization:** The PHP script validates field presence and sanitizes data against Cross-Site Scripting (XSS) and SQL injection using functions like `htmlspecialchars()`, `trim()`, and `stripslashes()`.
5. **Server Execution & Response:** PHP executes business logic (e.g., MySQL INSERT) and returns formatted confirmation output to the client.

3. Comparison Matrix: GET vs POST Method

Criterion	GET Method	POST Method
Data Visibility	Appended directly to URL (e.g., `?name=Nikhil&sem=5`).	Hidden inside HTTP request body.
Security Level	Low; unsuited for passwords or sensitive tokens.	Higher; parameters not exposed in history or logs.
Payload Capacity	Restricted (~2048 characters by browser limits).	Virtually unlimited (governed by `post_max_size`).
Browser Caching	Can be cached, bookmarked, and reloaded safely.	Cannot be bookmarked; browser prompts on re-submission.
Primary Use Case	Search engines, resource queries, page filtering.	Login credentials, payments, registrations, file uploads.

4. Complete Working Code (HTML Form + PHP Processing)

```
<?php
  $name = $email = $course = $successMsg = "";
  $nameErr = $emailErr = "";

  // Process form only when HTTP method is POST
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Validate and sanitize Name
    if (empty($_POST["student_name"])) {
      $nameErr = "Student name is required.";
    } else {
      $name = htmlspecialchars(trim($_POST["student_name"]));
    }

    // Validate and sanitize Email
    if (empty($_POST["student_email"])) {
      $emailErr = "Email address is required.";
    } else {
      $email = htmlspecialchars(trim($_POST["student_email"]));
    }

    $course = htmlspecialchars(trim($_POST["course"]));

    if (empty($nameErr) && empty($emailErr)) {
      $successMsg = "Registration Successful for " . $name . " (" . $course . ")!";
    }
  }
?>

<!DOCTYPE html>
<html>
<head>
  <title>PHP Form Handling</title>
  <style>
    .error { color: red; font-size: 13px; }
    .success { color: green; font-weight: bold; }
    form { max-width: 400px; padding: 20px; border: 1px solid #cbd5e1; border-radius: 6px; }
    input, select { width: 100%; padding: 8px; margin: 6px 0 12px 0; box-sizing: border-box; }
  </style>
</head>
<body>
  <h2>Student Admission Form</h2>

  <?php if ($successMsg): ?>
    <p class="success"><?php echo $successMsg; ?></p>
  <?php endif; ?>

  <form method="POST" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>">
    <label>Student Name:</label>
    <input type="text" name="student_name" value="<?php echo $name; ?>">
    <span class="error"><?php echo $nameErr; ?></span>

    <label>Email Address:</label>
    <input type="email" name="student_email" value="<?php echo $email; ?>">
    <span class="error"><?php echo $emailErr; ?></span>

    <label>Select Course:</label>
    <select name="course">
      <option value="BCA">BCA</option>
      <option value="B.Sc CS">B.Sc CS</option>
      <option value="B.Tech">B.Tech</option>
```

```
    </select>

    <input type="submit" value="Register Now">
  </form>
</body>
</html>
```

Q23. (a) Explain the while and do...while loops in JavaScript with examples. [4 Marks]

1. Core Concept & Distinction

Both `while` and `do...while` are iterative control flow statements used to execute a block of code repeatedly while a specified condition evaluates to true.

- **while Loop (Entry-Controlled):** The test condition is checked **before** loop execution. If the condition is false initially, the loop body executes **zero times**.
- **do...while Loop (Exit-Controlled):** The loop body executes **first**, and the condition is evaluated **afterwards**. It guarantees execution **at least once**, even if the condition is false from the start.

2. Syntax & Comparison Table

Feature	while Loop (Entry-Controlled)	do...while Loop (Exit-Controlled)
Condition Check	Evaluated at the start before entering loop.	Evaluated at the end after executing body.
Minimum Iterations	0 (Zero) times if condition is initially false.	1 (At least once) guaranteed.
Semicolon Syntax	No semicolon after loop header: `while() { }`	Requires semicolon after condition: `do { } while();`

3. Working Code Example

```
// Scenario: Initial condition is FALSE (count = 10, condition: count < 5)

let count = 10;

// while loop: Executes 0 times
console.log("Starting while loop:");
while (count < 5) {
  console.log("while loop executed: " + count);
  count++;
}
// Output: (Nothing logged)

// do...while loop: Executes 1 time guaranteed
console.log("Starting do...while loop:");
do {
  console.log("do...while executed once: " + count);
  count++;
} while (count < 5);
// Output: "do...while executed once: 10"
```

Q23. (b) Explain the creation, naming, and accessing of constants in PHP. [4 Marks]

1. Core Concept

A **constant** in PHP is an identifier (name) for a simple, immutable value that cannot be changed, reassigned, or undefined during script execution. Unlike variables, constants are automatically **global** across the entire script, including inside all functions.

2. Creation of Constants in PHP

PHP provides two standard mechanisms to create constants:

- **1. Using define() function:** Runtime definition function. Can be called conditionally inside if-statements or loops. `define("SITE_NAME", "BCA Academic Portal");`
- **2. Using const keyword:** Compile-time construct. Slightly faster; widely used inside class definitions and namespaces. `const DB_PORT = 3306;`

3. Naming Rules for Constants

- By convention, constants are written in uppercase letters with underscores (e.g., `MAX_USERS`, `DB_HOST`).
- Must begin with a letter or underscore (`_`), followed by any number of letters, numbers, or underscores.
- **No Dollar Sign:** Constants do **NOT** use the leading dollar sign (`$`) prefix used by PHP variables.

4. Accessing Constants & Global Scope Demonstration

```
<?php
  // Defining constants
  define("UNIVERSITY", "Bangalore City University");
  const MAX_ATTEMPTS = 3;

  function displayConfig() {
    // Constants are automatically available inside local function scope
    // No 'global' keyword required
    echo "University: " . UNIVERSITY . "<br>";
    echo "Max Login Attempts: " . MAX_ATTEMPTS . "<br>";
  }

  // Invoking function
  displayConfig();

  // Accessing constant dynamically using constant() function
  $constName = "UNIVERSITY";
  echo "Dynamic Access: " . constant($constName) . "<br>";

  // Attempting to reassign will cause a fatal error:
  // UNIVERSITY = "New University"; // Fatal Error: Cannot reassign
?>
```

MODEL QUESTION PAPER – 2

Q17. Explain how JavaScript changes element content, HTML attributes, and CSS styles through the DOM.

1. Core Concept & DOM Tree Model:

The **Document Object Model (DOM)** is a standardized in-memory programming interface provided by web browsers. It models an HTML document as a hierarchical tree of object nodes. JavaScript interacts with this tree to dynamically select elements, read or modify text/HTML contents, update attributes, and alter CSS styling in real time without reloading the webpage.

- **Selecting Elements:** Using `document.getElementById("id")`, `document.querySelector(".class/#id")`, or `document.querySelectorAll("selector")`.

2. Mechanisms for Modifying Content, Attributes, and Styles:

- **Modifying Element Content:**
  - `innerHTML`: Parses, reads, or replaces both plain text and child HTML markup inside an element.
  - `textContent`: Sets or returns raw textual content, ignoring and stripping HTML markup for enhanced security against XSS.

- **Modifying HTML Attributes:**
  - `setAttribute(name, value)`: Dynamically assigns or updates attributes (e.g., `src`, `href`, `alt`).
  - `getAttribute(name)` & `removeAttribute(name)`: Retrieves or deletes attributes.
  - Direct property assignment: e.g., `element.src = "new.jpg"` or `element.disabled = true`.
- **Modifying CSS Styles:**
  - Inline Style Object (`element.style.propertyName`): Modifies inline CSS properties using camelCase naming (e.g., `backgroundColor`, `fontSize`).
  - Class List API (`element.classList`): Dynamically toggles CSS classes via `classList.add()`, `classList.remove()`, and `classList.toggle()`.

3. Complete Interactive Working Code Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>DOM Modification Demo</title>
  <style>
    .base-card {
      padding: 16px;
      border: 1px solid #cbd5e1;
      width: 320px;
      border-radius: 6px;
      background-color: #f8fafc;
    }
    .active-mode {
      background-color: #dbeafe !important;
      border-color: #2563eb !important;
      color: #1e40af !important;
    }
  </style>
</head>
<body>

  <div id="profileCard" class="base-card">
    <h3 id="cardTitle">Guest User</h3>
    <p id="cardDesc">Welcome to the portal. Please log in.</p>
    <img id="avatarImg" src="guest.png" alt="Guest Avatar" width="100" height="100"><br><br>
    <button onclick="updateProfile()">Update Profile via DOM</button>
  </div>

  <script>
    function updateProfile() {
      // 1. Selecting elements
      let title = document.getElementById("cardTitle");
      let desc  = document.getElementById("cardDesc");
      let img   = document.getElementById("avatarImg");
      let card  = document.getElementById("profileCard");

      // 2. Modifying Content
      title.innerHTML = "Welcome, <em>Nikhil P</em>";
      desc.textContent = "Status: Authenticated Student (BCA 5th Sem).";

      // 3. Modifying Attributes
      img.setAttribute("src", "student.png");
      img.setAttribute("alt", "Verified Student Profile");

      // 4. Modifying CSS Styles
      card.style.borderColor = "#2563eb";
      card.style.boxShadow = "0 4px 6px rgba(0,0,0,0.1)";
      card.classList.toggle("active-mode");
    }
  </script>

</body>
</html>
```

4. DOM Modification Methods Summary Table:

Operation	Property / Method	Syntax Example	Key Characteristic
Change HTML Content	`innerHTML`	`el.innerHTML = "<b>Text</b>";`	Parses HTML strings into living child DOM elements.
Change Plain Text	`textContent`	`el.textContent = "New text";`	Safe against code injection; sets raw text only.
Update Attribute	`setAttribute()`	`el.setAttribute("src", "pic.jpg");`	Modifies or creates HTML attribute values.
Modify Inline Style	`style.property`	`el.style.backgroundColor = "gold";`	Uses camelCase notation; applies with highest specificity.
Toggle CSS Class	`classList.toggle()`	`el.classList.toggle("highlight");`	Adds class if absent; removes class if present.

5. Execution Output:

Before Click: Header displays "Guest User", description displays "Welcome to the portal", image points to "guest.png", card has gray border.
After Click: Header updates to "Welcome, *Nikhil P*", description shows verified status, image switches to "student.png", and card turns light blue with a soft shadow.

Q18. Explain the working of AJAX using the XMLHttpRequest object.

1. Core Concept:

AJAX (Asynchronous JavaScript and XML) is a client-side programming technique that allows web applications to communicate asynchronously with a web server in the background. It transmits and retrieves data without disrupting the user interface or triggering a full page reload, providing a desktop-like user experience.

2. AJAX Architectural Workflow:

```
[Client Browser: User Event (Click)]
      │
      ▼
[Create XMLHttpRequest Instance: new XMLHttpRequest()]
      │
      ▼
[Configure Request: xhr.open("GET", "get_data.php", true)]
      │
      ▼
[Dispatch Request to Web Server: xhr.send()]
      │ ─── HTTP GET / POST ──> [Web Server & Database]
      │                                  │
      │ <─── HTTP Response (JSON/Text) ──┘
      ▼
[Event Handler: xhr.onreadystatechange fires (readyState 0 -> 4)]
      │
      ▼
[Condition Met: readyState === 4 && status === 200]
      │
      ▼
[Parse responseText / JSON and Update Target DOM Element]
```

3. Lifecycle States & Key Properties of XMLHttpRequest:

- **The 5 Lifecycle ReadyStates (`readyState` property):**
  - `0 (UNSENT):` Client object created; `open()` has not yet been invoked.
  - `1 (OPENED):` `open()` has been called; HTTP method and destination URL are established.
  - `2 (HEADERS_RECEIVED):` `send()` has been called; response headers and HTTP status code are available.
  - `3 (LOADING):` Response payload is currently streaming/downloading from the server.
  - `4 (DONE):` Data transfer operation is completely finished; full response is ready for consumption.
- **Status Codes (`status` property):** Evaluates HTTP server response status, where `200` signifies Success and `404` denotes Resource Not Found.
- **Response Payloads:** `responseText` (returns raw string response) and `responseXML` (returns XML document object).
- **Core Methods:**
  - `open(method, url, async):` Initializes the HTTP request type, target script, and asynchronous boolean flag (default `true`).
  - `send(data):` Transmits the HTTP request to the server (with optional payload for POST).
  - `setRequestHeader(header, value):` Sets custom HTTP request headers (e.g., MIME types).

4. Complete Working Client-Side Code Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>AJAX XMLHttpRequest Demo</title>
</head>
<body>

  <h2>Live Student Details Lookup via AJAX</h2>
  <button id="fetchBtn" onclick="loadStudentData()">Fetch Student Details</button>

  <div id="displayArea" style="margin-top:15px; padding:12px; border:1px solid #ccc; width:300px;">
    Student details will load here asynchronously...
  </div>

  <script>
    function loadStudentData() {
      // Step 1: Instantiate the XMLHttpRequest object
      let xhr = new XMLHttpRequest();

      // Step 2: Configure request parameters (Method, URL, Asynchronous flag)
      xhr.open("GET", "student_info.json", true);

      // Step 3: Register event handler to track state transitions
      xhr.onreadystatechange = function() {
        // Verify transaction is complete (4) and successful (200)
        if (xhr.readyState === 4) {
          if (xhr.status === 200) {
            // Step 4: Parse received response text into JSON
            let student = JSON.parse(xhr.responseText);

            // Step 5: Dynamically update target DOM container
            document.getElementById("displayArea").innerHTML =
              "<strong>Reg No:</strong> " + student.regNo + "<br>" +
              "<strong>Name:</strong> " + student.name + "<br>" +
              "<strong>Course:</strong> " + student.course + "<br>" +
              "<strong>Marks:</strong> " + student.marks + "%";
          } else {
            document.getElementById("displayArea").innerHTML =
              "<span style='color:red;'>Error loading data. Status: " + xhr.status + "</span>";
          }
        }
      };

      // Step 6: Dispatch the request to the server
      xhr.send();
    }
  </script>

</body>
</html>
```

5. Execution Output:

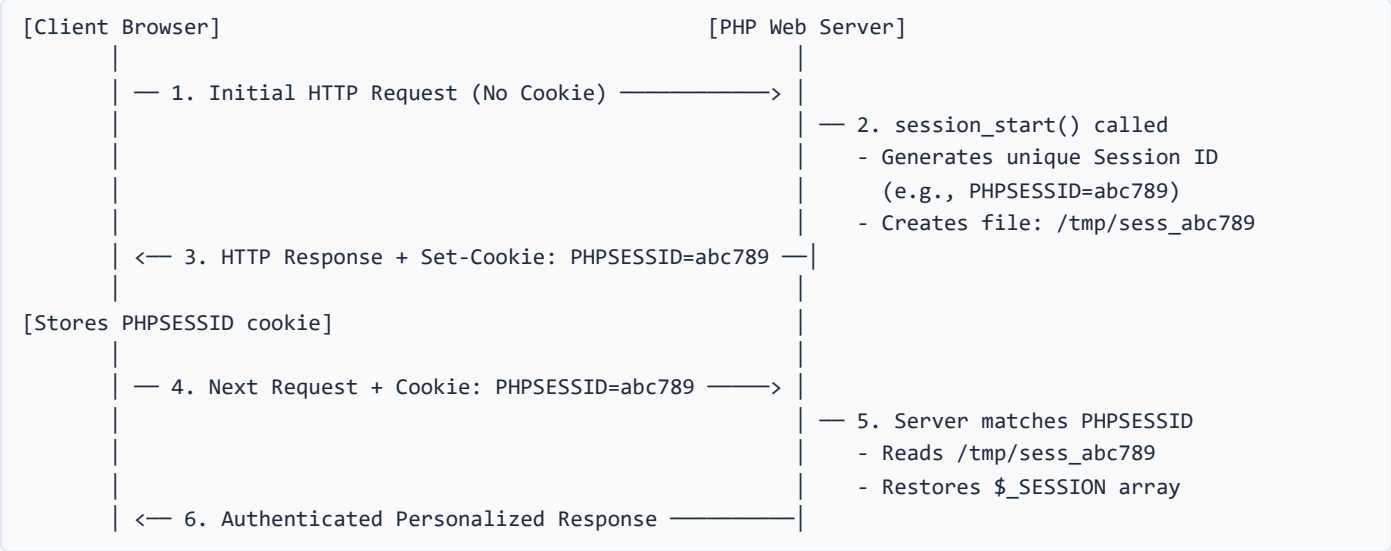
Initial State: "Student details will load here asynchronously..."
After Button Click: The browser fetches data seamlessly without reloading the page, updating the box to:
Reg No: U15CA21S001
Name: Nikhil P
Course: BCA 5th Semester
Marks: 92%

Q19. Explain the session-exchange mechanism and session-management functions in PHP.

1. Core Concept:

The **Hypertext Transfer Protocol (HTTP)** is inherently stateless, meaning the web server forgets each client connection immediately after serving a response. A **PHP Session** overcomes this limitation by persisting user-specific state and authentication credentials across multiple page requests on the server side using a unique session identifier.

2. Session Exchange & Handshake Mechanism:



3. Core PHP Session-Management Functions:

- **`session_start():`** Initializes a new session or resumes the existing session based on the session ID passed via cookie or GET. Must be invoked at the very top of the script before sending any HTML markup or whitespace.
- **`$_SESSION:`** Predefined superglobal associative array used to store and read session variables (e.g., `$_SESSION['user'] = "Nikhil";`).
- **`session_regenerate_id(true):`** Replaces the current session ID with a newly generated one while deleting old session data files, safeguarding against session fixation attacks.
- **`session_unset():`** Clears and unbinds all variables registered in the active `$_SESSION` array.
- **`session_destroy():`** Completely destroys all session data persisted in the physical storage medium on the web server.

4. Complete Multi-Page Working Code Example:

File 1: `login.php` (Session Initiation & Authentication):

```
<?php
// Start session before any output
session_start();

// Sample credential authentication
$username = "nikhil_p";
$role = "Administrator";

// Prevent session fixation
session_regenerate_id(true);

// Store data in session superglobal
$_SESSION['auth_user'] = $username;
$_SESSION['role']      = $role;
$_SESSION['login_time'] = time();

echo "<h3>Login Successful!</h3>";
echo "<p>Session established. <a href='dashboard.php'>Proceed to Dashboard</a></p>";
?>
```

File 2: `dashboard.php` (Session State Verification):

```
<?php
session_start();

// Verify user is authenticated
if (!isset($_SESSION['auth_user'])) {
    die("<p style='color:red;'>Access Denied: Please log in first.</p>");
}

echo "<h2>Welcome to User Dashboard</h2>";
echo "<p>Logged in as: <strong>" . htmlspecialchars($_SESSION['auth_user']) . "</strong></p>";
echo "<p>Role: <strong>" . htmlspecialchars($_SESSION['role']) . "</strong></p>";
echo "<p><a href='logout.php'>Log Out</a></p>";
?>
```

File 3: `logout.php` (Session Termination):

```
<?php
session_start();

// 1. Unset all session variables
$_SESSION = array();
session_unset();

// 2. Expire session cookie in client browser
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]
    );
}

// 3. Destroy server session file
session_destroy();

echo "<h3>Logged Out Successfully</h3>";
echo "<p>Session destroyed. <a href='login.php'>Login again</a></p>";
?>
```

5. Summary Reference Table:

Session Function	Purpose & Description	Execution Order
`session_start()`	Generates or resumes session; binds `$_SESSION`	First line of script
`session_regenerate_id()`	Replaces session ID to prevent hijacking	Upon successful login
`session_unset()`	Clears values in memory from `$_SESSION`	During logout cleanup
`session_destroy()`	Deletes server-side session file permanently	Final step of logout

---

Q20. Explain CRUD operations using the INSERT, SELECT, UPDATE, and DELETE commands in MySQL.

1. Core Concept:

CRUD is an acronym representing the four basic persistent data storage operations: **Create**, **Read**, **Update**, and **Delete**. In the MySQL relational database management system, these operations map directly to the four foundational Data Manipulation Language (DML) statements:

- **C (Create):** `INSERT` statement (adds new records).

- **R (Read):** `SELECT` statement (retrieves and queries records).
- **U (Update):** `UPDATE` statement (modifies existing records).
- **D (Delete):** `DELETE` statement (removes records).

2. Database Schema Setup & SQL Commands:

```
-- Setup Table Schema
CREATE TABLE students (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  email VARCHAR(50) UNIQUE,
  course VARCHAR(20),
  cgpa DECIMAL(3,2)
);

-- 1. CREATE: Insert record
INSERT INTO students (name, email, course, cgpa)
VALUES ('Nikhil P', 'nikhil@example.com', 'BCA', 9.20);

-- 2. READ: Query records with filters
SELECT id, name, course, cgpa
FROM students
WHERE course = 'BCA'
ORDER BY cgpa DESC;

-- 3. UPDATE: Modify existing record (Crucial WHERE clause)
UPDATE students
SET cgpa = 9.50, course = 'BCA Honours'
WHERE id = 1;

-- 4. DELETE: Remove record (Crucial WHERE clause)
DELETE FROM students
WHERE id = 1;
```

3. Complete PHP + MySQLi Integration Script:

```
<?php
// Database connection credentials
$host = "localhost";
$user = "root";
$pass = "";
$db   = "college_db";

// Establish MySQLi connection
$conn = new mysqli($host, $user, $pass, $db);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// 1. CREATE (INSERT)
$insert_sql = "INSERT INTO students (name, email, course, cgpa)
               VALUES ('Rahul Sharma', 'rahul@example.com', 'BCA', 8.80)";
if ($conn->query($insert_sql) === TRUE) {
    $new_id = $conn->insert_id;
    echo "Record inserted successfully. Generated ID: " . $new_id . "<br>";
}

// 2. READ (SELECT)
$select_sql = "SELECT id, name, course, cgpa FROM students WHERE course='BCA'";
$result = $conn->query($select_sql);
if ($result->num_rows > 0) {
    echo "<h4>Students List:</h4><ul>";
    while ($row = $result->fetch_assoc()) {
        echo "<li>" . $row['name'] . " - " . $row['course'] . " (CGPA: " . $row['cgpa'] . ")</li>";
    }
    echo "</ul>";
}

// 3. UPDATE
$update_sql = "UPDATE students SET cgpa = 9.10 WHERE id = " . $new_id;
$conn->query($update_sql);
echo "Student CGPA updated to 9.10.<br>";

// 4. DELETE
$delete_sql = "DELETE FROM students WHERE id = " . $new_id;
$conn->query($delete_sql);
echo "Student record deleted successfully.<br>";

// Close connection
$conn->close();
?>
```

4. CRUD Comparison & Safety Rules Table:

CRUD Operation	SQL Statement	Primary Purpose	Critical Safety Consideration
Create	`INSERT INTO`	Adds new rows to a table	Ensure primary key uniqueness and validate NOT NULL columns.
Read	`SELECT ... FROM`	Retrieves stored data matching criteria	Specify exact columns rather than `SELECT *` for performance.
Update	`UPDATE ... SET`	Modifies existing column values	Always include a specific `WHERE` clause; omitting it overwrites all rows.
Delete	`DELETE FROM`	Removes records from a table	Always include a specific `WHERE` clause; omitting it purges the entire table.

5. Execution Output:

Record inserted successfully. Generated ID: 2
Students List:
• Rahul Sharma - BCA (CGPA: 8.80)
Student CGPA updated to 9.10.
Student record deleted successfully.

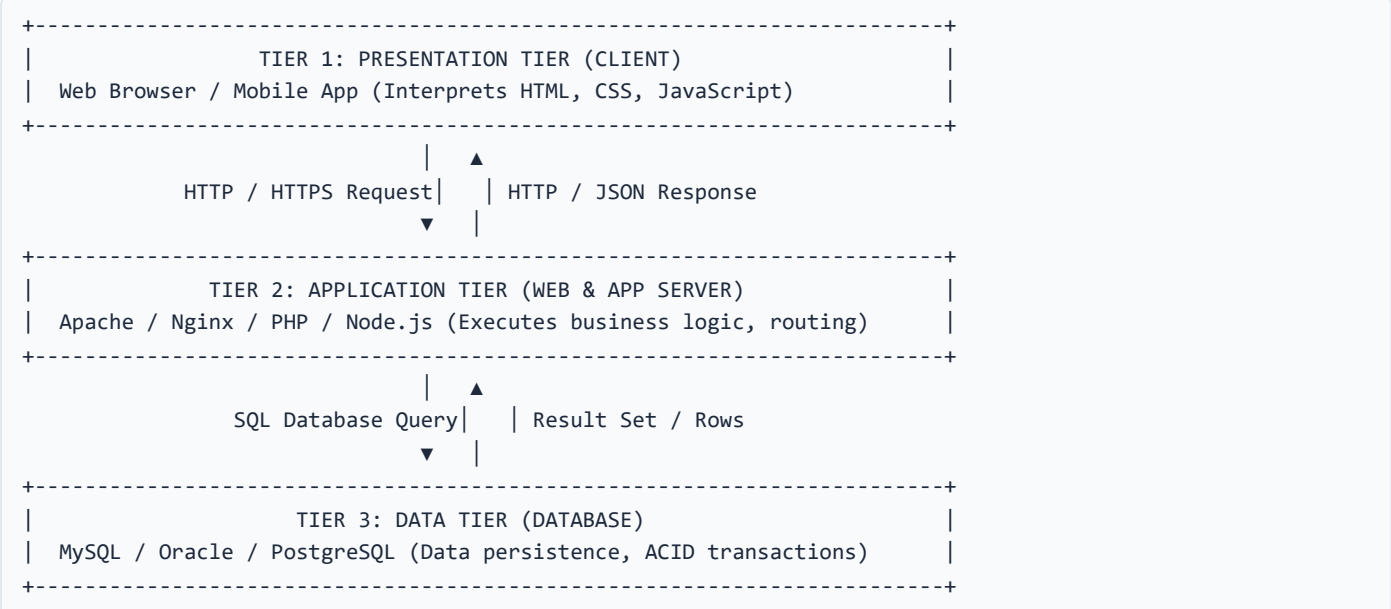
Q21. Explain client-server architecture with a neat diagram.

1. Core Concept:

Client-Server Architecture is a distributed computing paradigm where computing tasks, responsibilities, and workloads are partitioned between service requestors (**Clients**) and service providers (**Servers**). Communication between tiers takes place across a computer network using standardized communication protocols such as HTTP/HTTPS, TCP/IP, and SQL queries.

2. Architecture Diagrams:

Modern 3-Tier Web Architecture:



3. Detailed Explanation of Architectural Tiers:

- **Tier 1: Presentation Tier (Client):**
  - **Role:** Serves as the user-facing interface running on the user's local hardware (desktop, smartphone).
  - **Technologies:** Web browsers (Chrome, Firefox, Safari) executing HTML5, CSS3, and JavaScript.
  - **Responsibilities:** Renders visual components, handles UI events (clicks, keypresses), performs initial client-side form validation, and dispatches HTTP requests.
- **Tier 2: Application / Logic Tier (Web Server):**
  - **Role:** The middle tier that encapsulates the core functional business rules and logic of the system.
  - **Technologies:** Web servers (Apache, Nginx) paired with server-side runtimes (PHP, Python, Java).
  - **Responsibilities:** Authenticates user credentials, coordinates transactions, validates business constraints, processes session state, and queries the database.
- **Tier 3: Data Tier (Database Server):**
  - **Role:** Manages persistent, secure, and structured data storage.
  - **Technologies:** RDBMS engines (MySQL, PostgreSQL, Oracle).
  - **Responsibilities:** Guarantees ACID transactional consistency, performs data indexing, optimizes SQL query execution, and enforces data integrity constraints.

4. Comparison: 2-Tier vs. 3-Tier Architecture:

Evaluation Parameter	2-Tier Architecture (Client – Server)	3-Tier Architecture (Client – App – DB)
Architecture Model	Client communicates directly with Database.	Application server sits between Client and Database.
Scalability	Low; each client holds an open DB connection.	High; connection pooling and load balancing supported.
Security	Vulnerable; database credentials stored on client.	Robust; database is isolated behind an application firewall.
Maintenance	Difficult; client application updates required everywhere.	Simple; business logic modified centrally on app server.

5. Concrete Real-World Web Scenario:

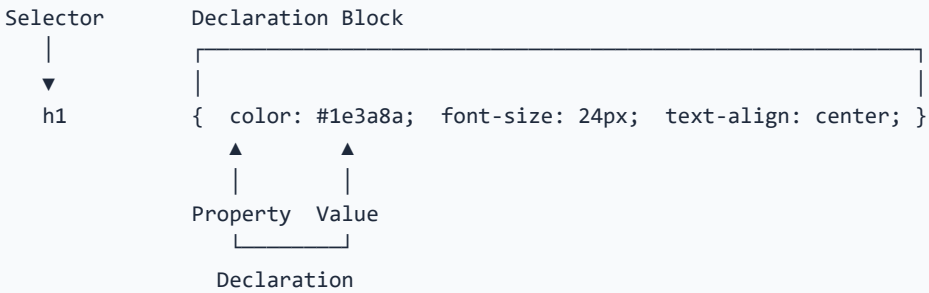
In an Online Banking Transfer: The user fills out an account form in their browser (Tier 1). The browser dispatches an HTTPS request to the banking application server (Tier 2), which verifies the account balance and executes business rules. The application server then initiates a secured SQL transaction to debit the funds in the core database (Tier 3), returning a confirmation screen to the client.

Q22. Explain the basic syntax and structure of a CSS rule with a suitable example.

1. Core Concept:

A **CSS Rule** (or *rule-set*) is the fundamental instructional unit in a Cascading Style Sheet that instructs the browser on how to visually style selected HTML elements. A complete CSS rule consists of two main parts: a **Selector** and a **Declaration Block**.

2. Anatomy Diagram of a CSS Rule:



3. Detailed Breakdown of Rule Components:

- **1. The Selector:**
  - Points to the HTML element(s) you wish to style.
  - Can target elements by tag name (`h1` , `p`), class name (`.card`), unique identifier (`#banner`), or attributes (`[type="text"]`).
- **2. The Declaration Block:**
  - Enclosed within opening `{` and closing `}` curly braces.
  - Contains one or more individual declarations separated by semicolons (`;`).
- **3. The Declaration:**
  - Composed of a **Property** and a **Value** separated by a colon (`:`).
  - Each declaration must terminate with a semicolon (`;`) to delimit instructions.
- **4. Property & Value:**
  - **Property:** The aesthetic characteristic being modified (e.g., `color` , `font-size` , `margin`).
  - **Value:** The specific stylistic setting assigned (e.g., `#1e3a8a` , `24px` , `center`).

4. Complete Illustrative Code Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Rule Structure Demo</title>
  <style>
    /* Rule 1: Element Selector */
    h1 {
      color: #0f172a;               /* Property: Value */
      font-family: Arial, sans-serif;
      text-align: center;
      border-bottom: 2px solid #0284c7;
    }

    /* Rule 2: Class Selector */
    .feature-card {
      background-color: #f0fdf4;
      border: 1px solid #22c55e;
      padding: 16px;
      border-radius: 8px;
      margin: 12px 0;
    }

    /* Rule 3: Grouped Selectors sharing declarations */
    h2, h3, p {
      line-height: 1.6;
      color: #334155;
    }
  </style>
</head>
<body>
  <h1>CSS Rule Syntax Demonstration</h1>
  <div class="feature-card">
    <h2>Modular Design</h2>
    <p>Every declaration maps a style property to a distinct value.</p>
  </div>
</body>
</html>
```

5. CSS Specificity and Hierarchy Summary Table:

Selector Type	Syntax Pattern	Matching Scope	Specificity Weight
Inline Style	`style="color: red;"`	Individual HTML tag	1000
ID Selector	`#mainHeader`	Single unique element per page	100
Class Selector	`.feature-card`	Reusable across multiple elements	10
Element Selector	`h1` , `p` , `div`	All elements of that HTML tag	1

Q23. (a) Explain any 4 semantic HTML elements with suitable examples. [4 Marks]
(b) Explain the color and background-color properties with suitable examples. [4 Marks]

(a) Four Semantic HTML Elements [4 Marks]:

Core Concept: Semantic HTML elements explicitly convey their structural meaning and intent to both the browser and developer, providing accessibility for screen readers and improving Search Engine Optimization (SEO), unlike generic non-semantic containers like `<div>` and `<span>` .

- 1. `<header>` : Represents introductory content or navigational aids, typically containing headings, company logos, search bars, or author information.
- 2. `<nav>` : Defines a dedicated section of the document intended exclusively for primary site navigation links.
- 3. `<article>` : Encloses self-contained, independent compositions that can be distributed or reused independently (e.g., blog posts, newspaper articles).
- 4. `<footer>` : Represents the concluding footer of an article or webpage, housing copyright notices, author credentials, and sitemap links.

Semantic HTML Code Example:

```
<header>
  <h1>Tech News Portal</h1>
  <nav>
    <a href="#home">Home</a> | <a href="#news">News</a> | <a href="#contact">Contact</a>
  </nav>
</header>

<article>
  <h2>Advances in Web Standards</h2>
  <p>Semantic markup improves accessibility for assistive technologies.</p>
</article>

<footer>
  <p>&copy; 2026 Academic Publishing. All rights reserved.</p>
</footer>
```

(b) The color and background-color Properties [4 Marks]:

Core Concept: In CSS, the `color` property defines the foreground color of textual content, while the `background-color` property sets the background fill color of an element's padding box.

Color Method	Syntax Format	Practical Example
Color Keyword	Predefined standard names	`color: navy;`
Hexadecimal (HEX)	`#RRGGBB` (6 hex digits)	`background-color: #1e3a8a;`
RGB Functional	`rgb(red, green, blue)`	`color: rgb(37, 99, 235);`
RGBA (with Opacity)	`rgba(r, g, b, alpha)`	`background-color: rgba(2, 132, 199, 0.25);`

Working CSS Code Example:

```
/* Applying color and background-color */
.callout-box {
  /* Foreground text color */
  color: #1e293b;

  /* Semi-transparent light-blue background */
  background-color: rgba(224, 242, 254, 0.85);

  padding: 16px;
  border-left: 4px solid #0284c7;
  border-radius: 4px;
}

.action-btn {
  color: #ffffff;             /* Crisp white text */
  background-color: #2563eb;  /* Solid blue background */
  border: none;
  padding: 8px 16px;
  border-radius: 4px;
}
```

MODEL QUESTION PAPER – 3

Q17. Explain how a web server works with a suitable diagram.

1. Core Concept and Definition

Definition: A web server is a specialized software application and underlying hardware system that stores website assets (HTML, CSS, JavaScript, media files) and delivers them to client web browsers across the Internet via standard network transfer protocols, primarily HTTP and HTTPS. It operates as a daemon process continuously listening for incoming client requests on standard network sockets: port 80 for unencrypted HTTP traffic and port 443 for encrypted TLS/SSL (HTTPS) communication.

2. Web Server Request-Response Lifecycle Diagram



3. Step-by-Step Server Execution Workflow

- **Socket Listening & Connection Acceptance:** The web server initializes a socket, binds to an assigned IP address and network port (80/443), and enters a passive listening state. When a client initiates a connection, the three-way TCP handshake completes, creating an active socket session.
- **HTTP Request Parsing:** The server reads the incoming raw byte stream, extracting the HTTP request method (e.g., `GET`, `POST`), requested Uniform Resource Identifier (URI), protocol version (e.g., `HTTP/1.1`, `HTTP/2`), and client headers (`Host`, `User-Agent`, `Cookie`).
- **Resource Routing and Access Verification:** The server checks directory permissions and configuration rules (such as `.htaccess` or Nginx location blocks). It verifies whether the requested resource exists on disk:
  - *Static Resource Handling:* For static files (`.html`, `.css`, `.png`), the server reads the file directly from storage and prepares the corresponding MIME type.
  - *Dynamic Script Processing:* For executable scripts (`.php`), the server invokes a server-side preprocessor (e.g., PHP interpreter via `mod_php` or PHP-FPM). The script executes, interacts with database servers (like MySQL), and returns dynamically generated HTML.

- **HTTP Response Formatting:** The server constructs an HTTP response packet consisting of an HTTP status line (e.g., `HTTP/1.1 200 OK` or `404 Not Found`), response headers (`Content-Type`, `Cache-Control`, `Set-Cookie`), and the payload entity body.
- **Transmission and Connection Management:** The payload is transmitted back to the client over the network. Under HTTP/1.1, connections remain open (`Keep-Alive`) to pipeline subsequent page assets without renegotiating TCP handshakes.

4. Leading Web Server Software

- **Apache HTTP Server:** An open-source, process- and thread-driven modular server supporting decentralized per-directory configuration via `.htaccess`.
- **Nginx:** An event-driven, asynchronous web server and reverse proxy renowned for high concurrency, low memory footprint, and rapid static asset caching.

Q18. Explain the important text-formatting tags in HTML with suitable examples.

1. Core Concept: Semantic vs Presentational Formatting

Definition: HTML text formatting tags modify the typographic appearance and semantic significance of text content. HTML standards classify text formatting elements into two primary architectural categories:

- **Physical (Presentational) Tags:** Dictate visual styling explicitly (e.g., bold or italic) without communicating underlying semantic significance to search engines or screen readers.
- **Logical (Semantic) Tags:** Convey informational importance, contextual emphasis, or grammatical role, allowing accessibility assistive technologies (like screen readers for visually impaired users) to alter speech inflection.

2. Comprehensive Text-Formatting Tags Reference Table

Tag	Classification	Semantic Meaning and Purpose	Default Browser Rendering
`<b>`	Physical	Draws visual attention to text without conveying extra importance or emphasis.	Renders text in **bold** typography.
`<strong>`	Logical (Semantic)	Indicates high semantic importance, seriousness, or urgency for enclosed text.	Renders text in **bold** typography.
`<i>`	Physical	Offsets text for stylistic reasons (technical terms, foreign words, thoughts).	Renders text in *italicized* typography.
`<em>`	Logical (Semantic)	Denotes stressed verbal emphasis that alters the linguistic meaning of a sentence.	Renders text in *italicized* typography.
`<u>`	Physical	Renders an unarticulated visual underline (often used for spelling error annotations).	Renders text with an underline.
`<mark>`	Logical (Semantic)	Indicates text highlighted for reference purposes due to relevance in another context.	Renders with a yellow highlight background.
`<small>`	Logical (Semantic)	Represents side-comments, legal disclaimers, copyrights, or fine print.	Renders text in a smaller font size.
`<del>`	Logical (Semantic)	Represents content that has been deleted or removed from a revised document.	Renders text with a ~~strikethrough~~ line.
`<ins>`	Logical (Semantic)	Represents content that has been newly added or inserted into a document.	Renders text with an underline.
`<sub>`	Physical	Renders subscript text positioned half a character below the normal baseline.	Lowers text: H_2O.
`<sup>`	Physical	Renders superscript text positioned half a character above the normal baseline.	Raises text: $x^2 + y^2$.

3. Complete Working HTML Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>HTML Text Formatting Demonstration</title>
</head>
<body>
  <h2>Demonstration of HTML Text Formatting Tags</h2>

  <!-- Physical vs Semantic Bold -->
  <p>Visual bold using b tag: <b>Product Catalog</b></p>
  <p>Semantic importance using strong: <strong>Warning: Disconnect power before service.</strong></p>

  <!-- Physical vs Semantic Italic -->
  <p>Foreign word using i tag: The term <i>ad hoc</i> means created for a specific purpose.</p>
  <p>Emphasis using em tag: You <em>must</em> submit your assignments on time.</p>

  <!-- Highlight and Small text -->
  <p>Search results: <mark>Web Programming</mark> is a core course in BCA.</p>
  <p><small>Copyright © 2026. All rights reserved.</small></p>

  <!-- Document Revisions -->
  <p>Discounted price: <del>&#8377;1200</del> <ins>&#8377;899</ins></p>

  <!-- Scientific and Mathematical Formulas -->
  <p>Chemical formula for sulfuric acid: H<sub>2</sub>SO<sub>4</sub></p>
  <p>Algebraic identity: (a + b)<sup>2</sup> = a<sup>2</sup> + 2ab + b<sup>2</sup></p>
</body>
</html>
```

4. Best Practices Summary

- Modern web development mandates utilizing logical tags (`<strong>` , `<em>`) instead of physical tags (`<b>` , `<i>`) whenever text carries informational emphasis, ensuring full accessibility compliance (WCAG 2.1).

Q19. Explain variable declaration and variable scope in JavaScript.

1. Core Concept of Variable Declarations

Definition: In JavaScript, a variable is a named symbolic reference to a storage location in memory. Prior to ECMAScript 2015 (ES6), variables were declared exclusively using the `var` keyword. ES6 introduced `let` and `const` to establish block-level scoping and immutable references, resolving common bugs associated with variable hoisting and unintended global leaks.

2. Comprehensive Comparison: var vs let vs const

Feature	var (Legacy ES5)	let (Modern ES6)	const (Modern ES6)
Scope Level	Function scope or Global scope. Does NOT respect block scope.	Block scope (confined within enclosing `{ ... }`).	Block scope (confined within enclosing `{ ... }`).
Re-declaration	Permitted within the same scope without syntax error.	Prohibited within the same scope (causes `SyntaxError`).	Prohibited within the same scope (causes `SyntaxError`).
Re-assignment	Permitted at any time.	Permitted at any time.	Prohibited; reference assignment is immutable.
Initialization	Optional at declaration (defaults to `undefined`).	Optional at declaration (defaults to `undefined`).	Mandatory at the moment of declaration.
Hoisting Behavior	Hoisted to scope top and automatically initialized to `undefined`.	Hoisted to scope top but remains uninitialized in Temporal Dead Zone (TDZ).	Hoisted to scope top but remains uninitialized in Temporal Dead Zone (TDZ).

3. The Three Types of Variable Scope in JavaScript

- **Global Scope:** Variables declared outside of any function or block boundary reside in the global execution context. They are accessible from any function, script, or callback in the application. In web browsers, global `var` declarations attach as properties of the `window` object.
- **Function (Local) Scope:** Variables declared within a function body are locally scoped to that function. They are instantiated upon function invocation and destroyed upon return; they cannot be accessed outside the enclosing function.
- **Block Scope:** Introduced in ES6 for `let` and `const`. Any code block bounded by curly braces `{ ... }` (such as `if` branches, `for` loops, or standalone blocks) defines a distinct scope boundary. Unlike `let` and `const`, `var` ignores block boundaries and leaks into the surrounding function or global environment.

4. Temporal Dead Zone (TDZ) and Hoisting Mechanism

Definition: Variable declarations are hoisted to the top of their respective scope by the JavaScript engine during the compile/parsing phase. While `var` is initialized with `undefined`, `let` and `const` are placed in a **Temporal Dead Zone (TDZ)** from the start of the block until the execution reaches the actual declaration statement. Accessing them inside the TDZ triggers a fatal `ReferenceError`.

5. Working JavaScript Demonstration Script

```
// 1. Demonstration of Global vs Function Scope
var globalVar = "I am global";

function testFunctionScope() {
    var functionScoped = "I am local to testFunctionScope";
    console.log(globalVar);       // Accessible: "I am global"
    console.log(functionScoped);  // Accessible: "I am local to testFunctionScope"
}
testFunctionScope();
// console.log(functionScoped);   // ReferenceError: functionScoped is not defined

// 2. Demonstration of Block Scope (var vs let)
if (true) {
    var leakedVar = "var leaks outside blocks!";
    let blockScoped = "let is strictly confined to block";
    const pi = 3.14159;
    // pi = 3.14;                 // TypeError: Assignment to constant variable.
    console.log(blockScoped);     // Output: "let is strictly confined to block"
}
console.log(leakedVar);           // Output: "var leaks outside blocks!"
// console.log(blockScoped);      // ReferenceError: blockScoped is not defined

// 3. Demonstration of Hoisting and TDZ
console.log(hoistedVar);          // Output: undefined (hoisted and initialized)
var hoistedVar = 100;

// console.log(tdzVar);           // ReferenceError: Cannot access 'tdzVar' before initialization
let tdzVar = 200;
```

Q20. What is form validation? Explain client-side and server-side form validation.

1. Core Concept and Definition

Definition: Form validation is the computational verification procedure that ensures user-submitted input adheres strictly to expected structural formats, data types, value boundaries, and security constraints before the data is processed, committed to a database, or passed to downstream application logic. Effective form validation protects databases from malformed records, prevents malicious code injection, and guides users in correcting formatting errors.

2. Comparative Architecture: Client-Side vs Server-Side Validation

Evaluation Dimension	Client-Side Validation	Server-Side Validation
Execution Location	Directly within the user's web browser prior to HTTP transmission.	On the remote backend server runtime (e.g., PHP interpreter).
Primary Objective	Instant user experience (UX) feedback, input formatting guidance.	System security, data integrity, attack prevention (SQLi, XSS).
Technologies Employed	HTML5 validation attributes, JavaScript DOM event listeners.	PHP server scripts, regular expressions, database query verification.
Security Reliability	Unreliable as a security barrier; easily bypassed by disabling JavaScript or using tools like cURL/Postman.	100% secure and authoritative; cannot be intercepted or modified by client manipulation.
Performance Impact	Saves network bandwidth by halting invalid requests locally.	Requires network round-trips and server compute cycles.

3. Client-Side Implementation (HTML5 and JavaScript)

Client-side validation provides real-time responsiveness using declarative HTML5 attributes (`required` , `type="email"` , `minlength` , `pattern`) supplemented by JavaScript logic to intercept form submission:

```
<!-- HTML Client-Side Form with JavaScript Validation -->
<form id="regForm" onsubmit="return validateClientForm()">
  <label for="user">Username (Min 5 chars):</label><br>
  <input type="text" id="user" name="user" required><br>
  <span id="userError" style="color:red;"></span><br>

  <label for="userEmail">Email Address:</label><br>
  <input type="email" id="userEmail" name="email" required><br><br>

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

<script>
function validateClientForm() {
    var userField = document.getElementById("user").value.trim();
    var errorSpan = document.getElementById("userError");

    // Check minimum length requirement
    if (userField.length < 5) {
        errorSpan.innerText = "Username must be at least 5 characters long.";
        return false; // Blocks HTTP request dispatch
    }
    errorSpan.innerText = "";
    return true; // Allows form submission
}
</script>
```

4. Server-Side Implementation (PHP Script)

Server-side validation acts as the definitive security gate, sanitizing raw inputs and validating data against server constraints regardless of client browser settings:

```php
<?php
// Server-Side Processing Script (process.php)
$errors = [];

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // 1. Validate Username presence and length
    if (empty($_POST["user"])) {
        $errors[] = "Username is mandatory.";
    } else {
        $user = trim($_POST["user"]);
        if (strlen($user) < 5) {
            $errors[] = "Username must be at least 5 characters.";
        }
    }

    // 2. Validate Email format using filter_var()
    if (empty($_POST["email"])) {
        $errors[] = "Email address is mandatory.";
    } else {
        $email = trim($_POST["email"]);
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $errors[] = "Invalid email format supplied.";
        }
    }

    // 3. Outcome Handling
    if (empty($errors)) {
        // Data is clean and verified: safe to persist in database
        $safeUser = htmlspecialchars($user);
        echo "Data validated successfully! Welcome, " . $safeUser;
    } else {
        // Output validation failures back to user
        foreach ($errors as $err) {
            echo "<p style='color:red;'>" . $err . "</p>";
        }
    }
}
?>
```

5. The Defense-in-Depth Principle

Modern web engineering adheres to the **Defense-in-Depth** security architecture: Client-side validation optimizes usability and eliminates latency for legitimate users, while server-side validation guarantees authoritative data security and system integrity.

Q21. Explain the different CSS positioning methods with suitable examples.

1. Core Concept of CSS Positioning

Definition: The CSS `position` property defines how an HTML element is situated on the page, whether it participates in normal document flow, and how it responds to the directional offset properties: `top`, `right`, `bottom`, and `left`, as well as 3D layering via `z-index`.

2. Detailed Breakdown of the Five Positioning Methods

- **position: static (Default):** Elements are rendered in natural document order according to standard block/inline flow. Directional offset properties (`top`, `bottom`, `left`, `right`) and the `z-index` property have no effect.
- **position: relative:** The element is initially placed in its normal position, then shifted relative to its original coordinate using offset properties. Crucially, its original physical space remains reserved in the layout flow, preventing surrounding elements from collapsing into its vacancy. It also establishes a containing positioning context for absolutely positioned descendant elements.

- **position: absolute:** The element is completely removed from normal document flow; no space is reserved for it. It is positioned relative to its closest ancestor that has a position other than `static`. If no positioned ancestor exists, it positions itself relative to the initial containing block (the `<html>` canvas).
- **position: fixed:** The element is removed from normal flow and positioned relative to the browser viewport window. It remains locked in its screen coordinates even as the user scrolls the document. Frequently used for persistent navigation bars and floating support buttons.
- **position: sticky:** A hybrid positioning mechanism. The element behaves as `relative` within normal document flow until the viewport scrolls to a designated offset threshold (e.g., `top: 0`), at which point it locks into place like `fixed` within the boundaries of its parent container.

3. Comprehensive Positioning Comparative Matrix

Position Value	Document Flow Status	Positioned Relative To	Offsets Effective?	Typical Real-World Use Case
`static`	Normal Flow	Natural flow hierarchy	No	Standard paragraphs, articles, default text.
`relative`	Normal Flow (Space reserved)	Its own normal default position	Yes	Minor positional tweaks; parent container for absolute items.
`absolute`	Removed from Flow	Nearest non-static ancestor	Yes	Modal dialogs, dropdown menus, notification badge overlays.
`fixed`	Removed from Flow	Browser Viewport window	Yes	Sticky top headers, floating back-to-top buttons.
`sticky`	Hybrid (Flow until threshold)	Scroll parent / Viewport	Yes	Table column headers, alphabetized list section headers.

4. Complete Working CSS & HTML Demonstration

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Positioning Demonstration</title>
  <style>
    /* 1. Fixed Header: pinned to viewport top */
    .site-header {
      position: fixed;
      top: 0;
      left: 0;
      width: 100%;
      height: 50px;
      background-color: #0f172a;
      color: #ffffff;
      line-height: 50px;
      padding-left: 20px;
      z-index: 1000;
    }

    body {
      padding-top: 60px; /* Offset to prevent fixed header overlap */
    }

    /* 2. Relative Container */
    .card-container {
      position: relative;
      width: 300px;
      height: 180px;
      border: 1px solid #cbd5e1;
      padding: 15px;
      margin: 20px 0;
      background-color: #f8fafc;
    }

    /* 3. Absolute Badge: positioned inside relative parent */
    .promo-badge {
      position: absolute;
      top: -10px;
      right: -10px;
      background-color: #dc2626;
      color: #ffffff;
      padding: 4px 8px;
      font-size: 11px;
      font-weight: 700;
      border-radius: 4px;
    }

    /* 4. Sticky Sidebar / Table Header */
    .sticky-bar {
      position: sticky;
      top: 50px; /* Sticks right below the fixed header */
      background-color: #e2e8f0;
      padding: 10px;
      font-weight: 600;
    }
  </style>
</head>
<body>
  <div class="site-header">Fixed Header Navigation</div>
```

```
  <div class="sticky-bar">Sticky Category Bar</div>

  <div class="card-container">
    <span class="promo-badge">NEW</span>
    <h4>Product Card</h4>
    <p>This container demonstrates absolute positioning inside a relative parent.</p>
  </div>
</body>
</html>
```

Q22. Explain the functions used to create, read, and write files in PHP.

1. Core Concept of File Handling in PHP

Definition: PHP incorporates a rich set of built-in filesystem functions allowing server-side scripts to interact directly with the server's local file storage. Files can be created, opened, read line-by-line or in bulk, written to, appended with new data, and closed securely to free operating system file handles.

2. File Access Modes in fopen()

The `fopen()` function binds a named file resource to a stream handle. Its behavior is dictated by the mode parameter:

Mode	Operation	File Pointer Position	File Creation / Truncation Behavior
`'r'`	Read Only	Beginning of file	Returns `false` with a warning if the file does not exist.
`'w'`	Write Only	Beginning of file	Truncates existing file to zero length; creates a new file if absent.
`'a'`	Append Only	End of file	Preserves existing data; creates a new file if absent.
`'r+'`	Read and Write	Beginning of file	Returns `false` if the file does not exist.
`'w+'`	Read and Write	Beginning of file	Truncates existing file to zero length; creates a new file if absent.
`'a+'`	Read and Append	End of file	Preserves existing data; creates a new file if absent.

3. Core File Manipulation Functions

- **fopen($filename, $mode):** Opens a file path or URL and returns an active file pointer resource, or `false` on failure.
- **fwrite($handle, $string):** Writes string contents to the open file pointer. Returns the number of bytes written.
- **fread($handle, $length):** Reads up to `$length` bytes from the current file pointer position.
- **fgets($handle):** Reads an entire line from the file handle until a newline character or EOF is encountered.
- **fgetc($handle):** Reads a single character from the open file handle.
- **feof($handle):** Tests whether the file pointer has reached the End-Of-File (EOF) marker.
- **fclose($handle):** Closes the active file resource handle, flushing output buffers to disk.
- **file_get_contents($filename):** High-level convenience function that reads an entire file into a string variable in a single call without manual handle management.
- **file_put_contents($filename, $data, FILE_APPEND):** Writes or appends string data directly to a file in a single operation.

4. Full Working PHP Demonstration Script

```php
<?php
// Script: File Creation, Writing, Appending, and Reading
$filePath = "activity_log.txt";

// 1. CREATE and WRITE data to file using 'w' mode
$fileHandle = fopen($filePath, "w");
if ($fileHandle) {
    $initialText = "Log Initialized: " . date("Y-m-d H:i:s") . "\n";
    fwrite($fileHandle, $initialText);
    fclose($fileHandle); // Always close open handles
    echo "File created and initial record written.<br>";
} else {
    die("Error: Unable to create file.");
}

// 2. APPEND additional data using 'a' mode
$fileHandle = fopen($filePath, "a");
if ($fileHandle) {
    $eventRecord = "User Admin logged in from IP: 192.168.1.10\n";
    fwrite($fileHandle, $eventRecord);
    fclose($fileHandle);
    echo "New log entry appended successfully.<br>";
}

// 3. READ file line-by-line using fgets() and feof()
echo "<h4>Reading File Contents Line by Line:</h4>";
$fileHandle = fopen($filePath, "r");
if ($fileHandle) {
    while (!feof($fileHandle)) {
        $line = fgets($fileHandle);
        echo htmlspecialchars($line) . "<br>";
    }
    fclose($fileHandle);
}

// 4. Reading entire file using high-level file_get_contents()
$allContent = file_get_contents($filePath);
echo "<h4>Reading Entire File with file_get_contents():</h4>";
echo "<pre>" . htmlspecialchars($allContent) . "</pre>";
?>
```

Q23. (a) Explain the advantages and disadvantages of CSS. (4 Marks)

1. Core Concept

Definition: Cascading Style Sheets (CSS) is the standard style sheet language used to describe the presentation, layout, and user interface formatting of documents written in HTML or XML.

2. Major Advantages of CSS

- **Separation of Presentation and Structure:** Keeps HTML files clean and semantic, while visual presentation rules are maintained independently.
- **Sitewide Consistency and Maintainability:** A single external stylesheet can dictate the look and feel of thousands of web pages simultaneously. Updating one CSS file updates the entire site.
- **Faster Page Load Speeds:** Browsers cache external CSS stylesheets locally after the initial page request, significantly reducing bandwidth consumption for subsequent visits.
- **Responsive Design Capabilities:** CSS Media Queries allow a single web document to fluidly adapt its layout across desktop monitors, tablets, and smartphone displays.

- **Superior Styling and Animation Controls:** Provides advanced visual features like Flexbox, Grid, gradients, shadows, transitions, and keyframe animations without requiring heavy JavaScript.

3. Major Disadvantages of CSS

- **Cross-Browser Rendering Inconsistencies:** Different browser rendering engines (Blink, Gecko, WebKit) occasionally interpret CSS properties or default margins differently, requiring browser vendor prefixes (e.g., `-webkit-`).
- **Cascading and Specificity Conflicts:** As stylesheets expand, conflicting selector rules and the cascading hierarchy can create specificity bugs that are difficult to debug in large codebases.
- **Lack of Built-in Security:** CSS does not provide programmatic access security; styling can be exploited in UI redress attacks (e.g., clickjacking or transparent overlay phishing).

Q23. (b) Write the differences between cookies and sessions in PHP. (4 Marks)

1. Core Concept

Definition: Because the HTTP protocol is completely stateless, web applications require state management mechanisms to persist user data across page requests. PHP achieves state tracking through **Cookies** (client-side storage) and **Sessions** (server-side storage).

2. Comprehensive Comparison Matrix: Cookies vs Sessions

Comparison Parameter	Cookies	Sessions
Storage Location	Stored on the client browser machine as small text files.	Stored on the web server's memory or temporary filesystem.
Security Level	Low security; vulnerable to client modification, tampering, and XSS theft.	High security; actual data stays on server; client only holds an opaque Session ID (`PHPSESSID`).
Storage Capacity	Limited to approximately 4 KB per individual cookie.	Virtually unlimited; bounded only by server physical storage and memory allocation.
Supported Data Types	Stores textual string data only.	Stores complex data structures (integers, strings, arrays, objects).
Network Bandwidth	Sent automatically within HTTP request headers on every page request, consuming bandwidth.	Only the small Session ID token is transmitted; payload data remains on the server.
Initialization in PHP	Created using `setcookie()` ; accessed via `$_COOKIE` superglobal array.	Initialized using `session_start()` ; accessed via `$_SESSION` superglobal array.
Typical Use Cases	"Remember Me" login tokens, tracking user preferences, UI theme selection (Dark/Light).	User authentication credentials, e-commerce shopping carts, multi-step transaction wizards.

MODEL QUESTION PAPER – 4

Q17. Explain any four CSS selectors with suitable examples.

1. Core Concept and Selector Architecture

CSS Selectors: Structural patterns and syntactic expressions defined within cascading stylesheets used to target, select, and bind specific HTML DOM elements to corresponding stylistic declarations. The browser's CSS parsing engine uses a mathematical scoring mechanism called **Specificity** (expressed as a four-component vector: Inline, ID, Class/Attribute/Pseudo-class, Element) to resolve cascade conflicts when multiple rules target the same element.

2. Detailed Explanation of Four Major Selectors

- **1. Element (Type) Selector:** Matches every HTML element that shares the specified tag name throughout the document tree. It operates at the baseline specificity level (`0,0,0,1`).
  Syntax: `p { line-height: 1.6; }` (targets all paragraph elements).

- **2. Class Selector:** Targets all HTML elements that possess a specific identifier inside their `class` attribute, indicated in CSS syntax by a preceding period (`.`). Classes can be applied across multiple heterogeneous tags and carry intermediate specificity (`0,0,1,0`).
  Syntax: `.highlight-card { background-color: #e0f2fe; }`
- **3. ID Selector:** Selects a single, unique HTML element bearing the matching value in its `id` attribute, denoted by a preceding hash symbol (`#`). Because an ID must be unique per HTML page, it commands high specificity (`0,1,0,0`).
  Syntax: `#main-header { border-bottom: 2px solid #0f172a; }`
- **4. Attribute Selector:** Selects elements based on the existence, exact match, or partial pattern match of their HTML attributes, enclosed within square brackets (`[...]`). Carries the same specificity weight as class selectors (`0,0,1,0`).
  Syntax: `input[type="submit"] { background-color: #16a34a; }` or `a[target="_blank"] { color: #dc2626; }`

3. Comprehensive Selectors Reference Table

Selector Type	Pattern Syntax	Specificity Weight	Targeting Scope	Standard Application
Element Selector	`elementName`	`0,0,0,1` (Lowest)	All instances of the specified HTML tag.	Base typographic styles and resets (`body` , `h1` , `p`).
Class Selector	`.className`	`0,0,1,0` (Medium)	Any element assigned that class name.	Reusable UI component patterns (buttons, cards, badges).
ID Selector	`#idName`	`0,1,0,0` (High)	Single unique element on the webpage.	Unique major layout landmarks (`#header` , `#footer`).
Attribute Selector	`[attr="val"]`	`0,0,1,0` (Medium)	Elements with matching attribute values.	Styling form inputs by type, links by target or protocol.

4. Full Working HTML & CSS Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Four CSS Selectors Demonstration</title>
  <style>
    /* 1. Element Selector */
    h2 {
      color: #1e293b;
      font-family: Arial, sans-serif;
      margin-bottom: 10px;
    }

    /* 2. Class Selector */
    .feature-card {
      background-color: #f8fafc;
      border: 1px solid #cbd5e1;
      border-radius: 6px;
      padding: 16px;
      margin-bottom: 12px;
    }

    /* 3. ID Selector */
    #primary-announcement {
      border-left: 5px solid #2563eb;
      background-color: #eff6ff;
    }

    /* 4. Attribute Selector */
    input[type="text"] {
      width: 280px;
      padding: 8px 10px;
      border: 1px solid #94a3b8;
      border-radius: 4px;
    }
    input[type="submit"] {
      padding: 8px 16px;
      background-color: #059669;
      color: #ffffff;
      border: none;
      border-radius: 4px;
      cursor: pointer;
    }
  </style>
</head>
<body>

  <h2>CSS Selectors Demonstration</h2>

  <!-- Applying Class and ID Selectors -->
  <div class="feature-card" id="primary-announcement">
    <p>This box is styled via both the .feature-card class and #primary-announcement ID.</p>
  </div>

  <!-- Applying Attribute Selectors -->
  <form action="#" method="get">
    <label for="username">User Name:</label>
    <input type="text" id="username" name="username" placeholder="Enter username">
    <input type="submit" value="Register">
  </form>
```

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

5. Rendered Output Summary

The `h2` title renders in dark slate Arial font via the element selector. The announcement container receives a soft blue background and solid blue left border through the class and ID selectors. The text input receives distinct padding and borders, while the submit button displays in emerald green via attribute selectors.

Q18. Explain responsive web design and the use of media queries. Also discuss how Flexbox and Grid help in creating responsive layouts.

1. Core Concept of Responsive Web Design (RWD)

Responsive Web Design (RWD): An architectural web development methodology ensuring that web applications dynamically adjust, reflow, and optimize their visual layout, typography, and interactive components across heterogeneous screen viewports (smartphones, tablets, laptops, and wide desktop displays) from a single unified codebase.

Three Core Pillars of RWD:

- **Fluid Grid Layouts:** Proportion-based spatial layouts calculated using percentages (`%`) or fractional units (`fr`) instead of fixed pixel dimensions.
- **Flexible Media:** Images and embedded video containers set with `max-width: 100%; height: auto;` to prevent viewport overflow.
- **CSS3 Media Queries:** Conditional styling rules that execute specific stylesheet blocks depending on device screen parameters.
- **Viewport Meta Tag:** Mandatory HTML head directive instructing mobile browsers to render pages at the device's true physical screen width rather than downscaling a virtual 980px desktop canvas: `<meta name="viewport" content="width=device-width, initial-scale=1.0">`

2. Media Queries Architecture and Breakpoints

Media queries interrogate device characteristics using the `@media` rule. Standard syntax checks media type and media expressions:

```
@media screen and (max-width: 768px) {
  /* Applied only on viewports narrower than 768px */
}
```

Standard Breakpoints: Mobile devices (`< 600px`), Tablets (`600px – 900px`), Desktop computers (`> 900px`).

3. How Flexbox and Grid Create Responsive Layouts

- **Flexbox (One-Dimensional Responsive Flow):** Enables responsive row-to-column shifting and fluid wrapping. Applying `flex-wrap: wrap;` causes overflowing child items to gracefully drop to subsequent lines. Setting `flex: 1 1 250px;` allows flex items to grow and shrink proportionally with an ideal base size of 250px.
- **CSS Grid (Two-Dimensional Responsive Matrices):** Facilitates intrinsic responsive layouts without requiring explicit media query breakpoints through the combination of `repeat()` , `auto-fit` , and `minmax()` functions: `grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));` This automatically determines the optimal number of columns per row based on available screen width.

4. Full Working Multi-Device Responsive Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Responsive Web Design Demo</title>
  <style>
    * {
      box-sizing: border-box;
      margin: 0;
      padding: 0;
    }
    body {
      font-family: Arial, sans-serif;
      padding: 16px;
    }

    /* Responsive Flexbox Navigation */
    .navbar {
      display: flex;
      justify-content: space-between;
      align-items: center;
      background-color: #1e293b;
      padding: 12px 20px;
      color: #ffffff;
    }
    .nav-links {
      display: flex;
      list-style: none;
      gap: 16px;
    }

    /* Responsive CSS Grid Container */
    .grid-container {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(240px, 1fr));
      gap: 16px;
      margin-top: 20px;
    }
    .card {
      background-color: #f1f5f9;
      border: 1px solid #cbd5e1;
      border-radius: 6px;
      padding: 20px;
      text-align: center;
    }

    /* Media Query for Mobile Handsets */
    @media screen and (max-width: 600px) {
      .navbar {
        flex-direction: column;
        align-items: flex-start;
        gap: 10px;
      }
      .nav-links {
        flex-direction: column;
        width: 100%;
      }
      .card {
        background-color: #e2e8f0;
```

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

  <nav class="navbar">
    <h3>Responsive Portal</h3>
    <ul class="nav-links">
      <li>Home</li>
      <li>Courses</li>
      <li>About</li>
      <li>Contact</li>
    </ul>
  </nav>

  <section class="grid-container">
    <div class="card"><h4>Module 1: HTML5</h4><p>Semantic Web Layouts</p></div>
    <div class="card"><h4>Module 2: CSS3</h4><p>Flexbox and Grid Design</p></div>
    <div class="card"><h4>Module 3: JavaScript</h4><p>DOM and Event Handling</p></div>
    <div class="card"><h4>Module 4: PHP & MySQL</h4><p>Server-Side Data Backend</p></div>
  </section>

</body>
</html>
```

5. Layout Behavior Across Viewports

Device Viewport	Navbar Behavior (Flexbox)	Grid Container Columns	Visual Effect
Desktop (> 900px)	Horizontal row layout with space-between.	4 equal columns side-by-side.	Full panoramic desktop workspace.
Tablet (600px – 900px)	Horizontal compact row layout.	2 columns per row automatically reflowing.	Balanced multi-column layout.
Mobile (< 600px)	Vertical stacked column layout.	1 single column filling full width.	Touch-friendly vertical feed with zero horizontal scrolling.

Q19. Explain how to create, read, update, and delete cookies in PHP.

1. Core Concept of Cookies in PHP

HTTP Cookie: A compact text file (limited to 4 KB) created by the server and stored locally on the client's web browser. Cookies enable stateless HTTP servers to maintain persistent user state, track sessions, store theme preferences, and remember user identity between distinct browsing requests.

2. Full Parameter Signature of setcookie()

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

- **name:** The identifier string of the cookie (e.g., `"username"`).
- **value:** The text string stored within the cookie.
- **expire:** Unix timestamp indicating when the cookie becomes invalid (e.g., `time() + 86400` for 1 day). Omission creates a temporary session cookie.
- **path:** Server directory where cookie is active (usually `"/"` for site-wide availability).
- **domain:** The domain scope where the cookie is visible (e.g., `"example.com"`).
- **secure:** Boolean flag; if `true`, transmits the cookie exclusively across secure HTTPS connections.
- **httponly:** Boolean flag; if `true`, blocks client JavaScript access via `document.cookie`, preventing Cross-Site Scripting (XSS) attacks.

3. Complete CRUD Lifecycle Operations

- **Create (Set Cookie):** Created using `setcookie()` prior to emitting any HTML headers or content: `setcookie("studentName", "Nikhil", time() + 3600, "/");`
- **Read (Retrieve Cookie):** Accessed on subsequent requests via the PHP associative superglobal array `$_COOKIE`: `$name = isset($_COOKIE["studentName"]) ? $_COOKIE["studentName"] : "Guest";`
- **Update (Modify Cookie):** Overwritten by calling `setcookie()` with the identical name and path parameters, supplying the new value: `setcookie("studentName", "Nikhil Kumar", time() + 3600, "/");`
- **Delete (Remove Cookie):** Removed by calling `setcookie()` with an expiration timestamp explicitly set in the past, followed by unsetting the superglobal variable: `setcookie("studentName", "", time() - 3600, "/"); unset($_COOKIE["studentName"]);`

4. Full Working PHP Cookie Manager Script

```
<?php
// Notice: Cookie operations MUST precede any HTML output or whitespace!

// 1. CREATE a new cookie valid for 7 days
$cookieName = "preferredLanguage";
$cookieValue = "English";
$expireTime = time() + (86400 * 7); // 7 days from current time
setcookie($cookieName, $cookieValue, $expireTime, "/", "", false, true);

// 2. READ the existing cookie
$currentLanguage = "Not Set";
if (isset($_COOKIE[$cookieName])) {
    $currentLanguage = htmlspecialchars($_COOKIE[$cookieName]);
}

// 3. UPDATE the cookie with a new preference
if (isset($_GET['update']) && $_GET['update'] === 'kannada') {
    setcookie($cookieName, "Kannada", time() + (86400 * 7), "/", "", false, true);
    $_COOKIE[$cookieName] = "Kannada"; // Update local array for immediate rendering
    $currentLanguage = "Kannada";
}

// 4. DELETE the cookie if requested
if (isset($_GET['action']) && $_GET['action'] === 'delete') {
    // Expire the cookie 1 hour in the past
    setcookie($cookieName, "", time() - 3600, "/");
    unset($_COOKIE[$cookieName]);
    $currentLanguage = "Deleted";
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>PHP Cookie Management Lifecycle</title>
</head>
<body>
  <h2>PHP Cookie CRUD Operations</h2>
  <p>Active Language Preference: <strong><?php echo $currentLanguage; ?></strong></p>

  <p>
    <a href="?update=kannada">Update Language to Kannada</a> |
    <a href="?action=delete">Delete Cookie</a>
  </p>
</body>
</html>
```

5. Operations Summary Table

Operation	PHP Function / Superglobal	Expiration Parameter	Browser Action
Create	`setcookie("user", "val", time() + 3600, "/");`	Future timestamp (e.g. `+3600s`)	Saves cookie file into browser storage cache.
Read	`$val = $_COOKIE["user"];`	N/A (Read-only access)	Transmits cookie payload in HTTP request header.
Update	`setcookie("user", "newVal", time() + 3600, "/");`	Future timestamp with identical path	Overwrites existing cookie file with new value.
Delete	`setcookie("user", "", time() - 3600, "/");`	Past timestamp (e.g. `-3600s`)	Purges cookie file immediately from disk/memory.

Q20. Explain how to establish a connection between PHP and MySQL using the mysqli_connect() function.

1. Core Concept of Database Connectivity in PHP

Database Connectivity: Web applications depend on communication between server-side PHP scripts and relational database management systems like MySQL to execute persistent CRUD transactions. PHP provides the **MySQLi (MySQL Improved)** extension, featuring both procedural functions and object-oriented classes optimized for modern MySQL servers.

2. Function Signature & Parameters of mysqli_connect()

```
$conn = mysqli_connect(hostname, username, password, database_name, port, socket);
```

- **hostname:** The network address or domain name where the MySQL server resides (typically `"localhost"` or `"127.0.0.1"` for local development).
- **username:** The MySQL database user account name (e.g., `"root"`).
- **password:** The secret authentication password associated with the database user.
- **database_name:** (Optional) The specific schema database to select upon connection.
- **port:** (Optional) The TCP port number where MySQL listens (default is `3306`).
- **socket:** (Optional) The Unix socket or named pipe to use.

3. Connection Validation and Error Handling Functions

- **mysqli_connect_error():** Returns a clear textual description of the connection error if the attempt fails, or `null` if successful.
- **mysqli_connect_errno():** Returns the numerical error code associated with the failed connection attempt.
- **mysqli_close($conn):** Explicitly terminates the open MySQL database connection link and releases allocated socket resources.

4. Full Working PHP Database Connection Script

```php
<?php
// Database configuration credentials
$dbHost = "localhost";
$dbUser = "root";
$dbPass = "";          // Empty by default on standard local XAMPP/WAMP
$dbName = "bcu_student_db";

// 1. Establishing connection to MySQL database
$conn = mysqli_connect($dbHost, $dbUser, $dbPass, $dbName);

// 2. Verifying connection status and handling failure
if (!$conn) {
    // Terminate script execution and print error diagnostics
    die("Database Connection Failed: " . mysqli_connect_error() .
        " (Error Code: " . mysqli_connect_errno() . ")");
}

echo "Connection Successful! Connected to database: " . $dbName . "<br>";

// Setting UTF-8 character encoding
mysqli_set_charset($conn, "utf8mb4");

// 3. Executing a sample query to demonstrate active link
$sql = "SELECT 'Database Server Active' AS status_msg";
$result = mysqli_query($conn, $sql);

if ($result) {
    $row = mysqli_fetch_assoc($result);
    echo "Diagnostic Check: " . $row['status_msg'] . "<br>";
    // Free query memory
    mysqli_free_result($result);
}

// 4. Closing the database connection link
mysqli_close($conn);
echo "Connection closed successfully.";
?>
```

5. Procedural vs. Object-Oriented MySQLi Comparison

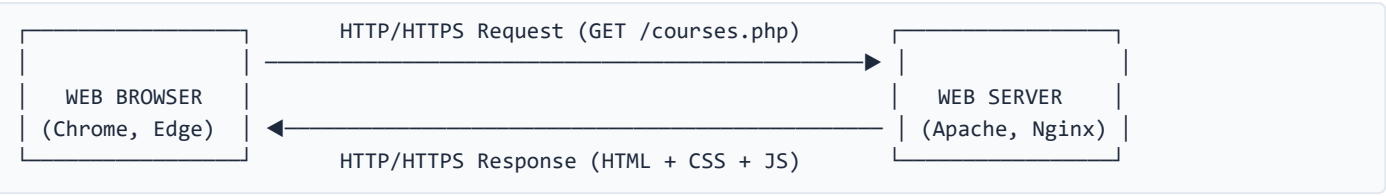
Feature	Procedural MySQLi	Object-Oriented MySQLi
Initiation	`$conn = mysqli_connect($h, $u, $p, $db);`	`$conn = new mysqli($h, $u, $p, $db);`
Error Checking	`if (!mysqli_connect_error())`	`if (!$conn->connect_error)`
Query Execution	`$res = mysqli_query($conn, $sql);`	`$res = $conn->query($sql);`
Closing Link	`mysqli_close($conn);`	`$conn->close();`
Programming Paradigm	Function-oriented; uses link resource parameters.	Class instance method calls on connection object.

Q21. Explain the features of a web browser and web server.

1. Core Concepts & Client-Server Paradigm

The World Wide Web functions through a distributed **Client-Server Architecture**. The **Web Browser** acts as the client-side user interface agent that requests, parses, and renders hypermedia documents. The **Web Server** acts as the remote daemon listening for network requests, executing server-side logic, and delivering requested files.

2. Client-Server Interaction Workflow



3. Key Features of a Web Browser

- **Layout and Rendering Engine:** Interprets HTML markup and CSS styling rules to construct the DOM (Document Object Model) and CSSOM trees, generating the visual layout pixels on screen (e.g., Blink in Chrome, Gecko in Firefox, WebKit in Safari).
- **JavaScript Interpreter / Engine:** High-performance runtime containing a Call Stack, Memory Heap, and Just-In-Time (JIT) compiler (e.g., Google V8, SpiderMonkey) to execute dynamic client-side scripts.
- **User Interface Components:** Standardized navigational tools including address bar (Omnibox), back/forward buttons, refresh button, tab management, and developer tools (F12 console/inspector).
- **Networking and Protocol Stack:** Implements HTTP, HTTPS, WebSocket, and DNS resolution to establish secure TCP/IP connections.
- **Client-Side Storage:** Provides localized sandboxed data persistence mechanisms including Cookies, `localStorage`, `sessionStorage`, and IndexedDB.
- **Security Sandbox:** Enforces the Same-Origin Policy (SOP), Content Security Policy (CSP), and SSL/TLS certificate validation to protect user data from cross-site scripts.

4. Key Features of a Web Server

- **HTTP/HTTPS Daemon Listener:** Continuously listens for incoming network connections on standard ports (Port 80 for HTTP, Port 443 for HTTPS) and manages concurrent requests.
- **Virtual Hosting:** Enables a single server machine to host multiple independent domain names and websites through IP-based or name-based virtual host configurations.
- **Server-Side Script Execution:** Integrates with language runtimes (such as PHP via FastCGI/PHP-FPM, Python, Node.js) to generate dynamic content prior to transmission.
- **Security and Access Control:** Enforces directory-level permissions, URL rewrites, firewall rules, and TLS handshake negotiations.
- **Comprehensive Logging and Monitoring:** Records all incoming web traffic into `access.log` and diagnostic errors into `error.log` for performance analytics.
- **Content Compression and Caching:** Compresses response payloads using Gzip or Brotli algorithms and handles HTTP caching headers (`ETag`, `Cache-Control`).

5. Comprehensive Comparison Matrix

Comparison Basis	Web Browser (Client)	Web Server (Server)
Primary Role	Requests, interprets, and renders web pages for users.	Accepts requests, processes logic, and serves files.
Execution Location	Runs on user end-point hardware (PC, phone, tablet).	Runs on remote host hardware / data center clouds.
Popular Examples	Google Chrome, Mozilla Firefox, Microsoft Edge, Safari.	Apache HTTP Server, Nginx, Microsoft IIS, LiteSpeed.
Supported Technologies	HTML5, CSS3, Client JavaScript, WebAssembly.	PHP, Python, Java, Node.js, SQL Databases.
Networking Role	Initiates HTTP requests; acts as TCP client.	Listens on ports 80/443; acts as TCP server daemon.
Data Storage	Cookies, localStorage, IndexedDB on user device.	Relational DBs (MySQL), file systems, server cache.

Q22. Explain JSON objects, arrays, and nested structures.

1. Core Concept and JSON Syntax Rules

JSON (JavaScript Object Notation): A lightweight, standardized, human-readable, and text-based data interchange format. Derived from JavaScript object literal notation, JSON is completely language-independent and universally supported across modern programming languages for client-server API data exchange.

Core Syntax Rules:

- Data is represented in key/value pairs.
- Keys must always be strings enclosed in double quotes (e.g., `"name"`).
- Data elements are separated by commas; trailing commas are strictly invalid.
- Permitted data types: String, Number, Boolean (`true` / `false`), `null`, Object, Array.
- Prohibited data types: Functions, Dates, `undefined`.

2. Detailed Breakdown of JSON Structures

- **JSON Object:** An unordered collection of key/value pairs bounded by curly braces `{...}`.
  Syntax: `{ "id": 101, "course": "BCA", "active": true }`
- **JSON Array:** An ordered sequence of values bounded by square brackets `[...]`, separated by commas and indexed starting at 0.
  Syntax: `[ "HTML", "CSS", "JavaScript", "PHP" ]`
- **Nested Structures:** Complex hierarchies where JSON objects contain arrays, arrays contain objects, or objects contain child objects. This models real-world enterprise entity-relationship data accurately.

3. Complete JSON Data Payload Example

```
{
  "university": "Bengaluru University",
  "collegeCode": "BCU501",
  "accredited": true,
  "student": {
    "regNo": "BCA202401",
    "name": "Nikhil Kumar",
    "semester": 5,
    "cgpa": 8.75,
    "address": {
      "city": "Bengaluru",
      "pincode": 560001
    },
    "subjects": [
      { "code": "DCCA511", "title": "Web Programming", "credits": 4 },
      { "code": "DCCA512", "title": "Data Analytics", "credits": 4 }
    ],
    "skills": ["HTML5", "CSS3", "JavaScript", "PHP", "MySQL"]
  }
}
```

4. JSON Parsing and Serialization in JavaScript & PHP

```javascript
// ========================================
// JavaScript JSON Methods
// ========================================
const jsonString = '{"course":"BCA","durationYears":3}';

// JSON.parse(): Converts JSON text string into a native JavaScript object
const jsObj = JSON.parse(jsonString);
console.log(jsObj.course); // Output: BCA

// JSON.stringify(): Serializes a JavaScript object into a JSON formatted string
const studentData = { name: "Ananya", marks: 95 };
const serializedJSON = JSON.stringify(studentData);
console.log(serializedJSON); // Output: {"name":"Ananya","marks":95}
```

```php
<?php
// ========================================
// PHP JSON Functions
// ========================================
$studentArray = [
    "name" => "Rahul",
    "semester" => 5,
    "grades" => ["A", "A+", "O"]
];

// json_encode(): Converts a PHP associative array into a JSON string
$jsonOutput = json_encode($studentArray);
echo $jsonOutput; // {"name":"Rahul","semester":5,"grades":["A","A+","O"]}

// json_decode(): Parses a JSON string back into a PHP associative array (true flag)
$parsedArray = json_decode($jsonOutput, true);
echo $parsedArray["name"]; // Rahul
?>
```

5. Structural Formats Summary

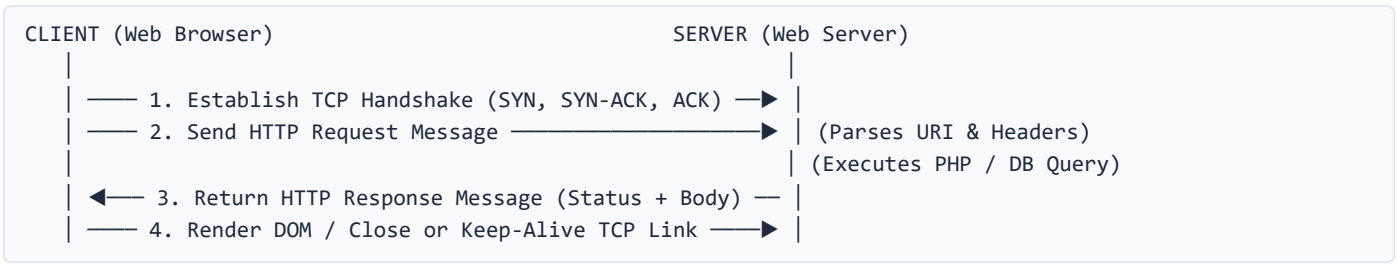
Structure	Delimiter	Indexing / Keying	Primary Use Case
JSON Object	Curly braces `{ }`	String keys (`"key": value`)	Representing single structured records or entities.
JSON Array	Square brackets `[ ]`	0-based numerical indexing	Representing lists, collections, or sequential data.
Nested Structure	Combination of `{ }` and `[ ]`	Multi-level hierarchical keys and indices	Representing relational datasets and API responses.

Q23. (a) Explain the HTTP request-response mechanism. [4 Marks]

1. Core Concept of HTTP Architecture

HTTP (Hypertext Transfer Protocol): An application-layer, stateless protocol operating over TCP/IP that serves as the communication foundation of the World Wide Web. It functions through a synchronous or asynchronous **Client-Server Request-Response Cycle** where a client transmits an HTTP Request message, and the server returns an HTTP Response message containing status codes and requested content.

2. HTTP Request-Response Workflow



3. Structure of an HTTP Request Message

- **Request Line:** Contains the HTTP verb, target URI path, and protocol version.
  `GET /courses/bca.html HTTP/1.1`
- **Request Headers:** Metadata key-value lines detailing client capabilities.
  `Host: bcu.ac.in`
  `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)`
  `Accept: text/html,application/xhtml+xml`
- **Empty Line:** Mandatory carriage return indicating the end of header lines.
- **Message Body:** Optional payload transmitted during `POST` or `PUT` submissions (e.g., form data or JSON).

4. Structure of an HTTP Response Message

- **Status Line:** Protocol version, 3-digit numerical status code, and reason phrase.
  `HTTP/1.1 200 OK` (or `404 Not Found`, `500 Server Error`)
- **Response Headers:** Server metadata properties.
  `Content-Type: text/html; charset=UTF-8`
  `Content-Length: 4280`
  `Server: Apache/2.4.52`
- **Empty Line:** Separates response headers from body payload.
- **Response Body:** The actual delivered resource payload (HTML page, JSON data, image binary).

Q23. (b) Explain setTimeout(), clearTimeout(), setInterval(), and clearInterval() with examples. [4 Marks]

1. Core Concept of JavaScript Timers

In JavaScript, timers are asynchronous APIs provided by the browser's `window` object (part of Web APIs) that allow developers to delay function execution or repeatedly execute code at specified time intervals (measured in milliseconds, where 1000 ms = 1 second).

2. Detailed Explanation of the Four Timer Methods

- **setTimeout(function, delay):** Executes a callback function once after a specified delay in milliseconds. Returns a unique numerical timer ID.
  Syntax: `const timerId = setTimeout(callbackFunction, 2000);`
- **clearTimeout(timerId):** Cancels a scheduled timeout before its callback triggers, utilizing the timer ID returned by `setTimeout()`.
  Syntax: `clearTimeout(timerId);`
- **setInterval(function, interval):** Repeatedly invokes a callback function at periodic intervals of specified milliseconds until explicitly halted.
  Syntax: `const intervalId = setInterval(callbackFunction, 1000);`
- **clearInterval(intervalId):** Stops and destroys a recurring interval timer previously established by `setInterval()`.
  Syntax: `clearInterval(intervalId);`

3. Full Working JavaScript Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Timer Methods Demonstration</title>
</head>
<body>

  <h3>Countdown Timer and Delayed Alert</h3>
  <p>Remaining Seconds: <strong id="countdownDisplay">5</strong></p>
  <button id="stopBtn">Stop Countdown</button>
  <p id="statusMsg"></p>

  <script>
    let count = 5;
    const display = document.getElementById("countdownDisplay");
    const status = document.getElementById("statusMsg");

    // 1. setInterval(): Counts down every 1000ms (1 second)
    const countdownTimer = setInterval(function() {
      count--;
      display.textContent = count;

      if (count <= 0) {
        // 2. clearInterval(): Stops interval when count hits 0
        clearInterval(countdownTimer);
        status.textContent = "Countdown Complete!";
      }
    }, 1000);

    // 3. clearTimeout(): Stop countdown button click listener
    document.getElementById("stopBtn").addEventListener("click", function() {
      clearInterval(countdownTimer);
      clearTimeout(delayedNotification);
      status.textContent = "Countdown and Notification Aborted by User.";
    });

    // 4. setTimeout(): Displays notification after 6000ms (6 seconds)
    const delayedNotification = setTimeout(function() {
      alert("Notice: 6 seconds have elapsed!");
    }, 6000);
  </script>

</body>
</html>
```

4. Methods Summary Table

Method Name	Execution Pattern	Returned Identifier	Cancellation Function
`setTimeout()`	Single delayed execution.	Timeout ID (positive integer)	`clearTimeout(id)`
`setInterval()`	Repeated recurring execution.	Interval ID (positive integer)	`clearInterval(id)`

MODEL QUESTION PAPER – 5

Q17. Explain the working of the World Wide Web with a suitable example.

1. Core Concept and Architecture

The World Wide Web (WWW) is a distributed, global information system where digital assets, documents, and multimedia services are identified by Uniform Resource Locators (URLs) and interconnected via hyperlinks. It operates on top of the physical Internet infrastructure using a **client-server architecture** governed primarily by the HyperText Transfer Protocol (HTTP/HTTPS).
The architecture involves three primary participants:

- **Web Client (Browser):** Sends requests for resources, interprets received payloads (HTML, CSS, JavaScript), and renders the graphical interface.
- **Domain Name System (DNS):** Acts as the distributed Internet phonebook, translating human-friendly domain names into machine-routable IP addresses.
- **Web Server:** Listens for incoming HTTP/HTTPS connection requests, executes server-side scripts (such as PHP), accesses relational databases, and returns HTTP response payloads.

2. Architectural Workflow Diagram

```
[ Web Browser (Client) ]
         |
         | 1. User enters URL: https://www.example.com/index.html
         v
[ DNS Resolution ] --------> Queries Root, TLD, Authoritative DNS
         |                  <-------- Returns Web Server IP (e.g. 93.184.216.34)
         v
[ TCP 3-Way Handshake ] ---> SYN -> SYN-ACK -> ACK (Port 80/443)
[ TLS / SSL Handshake ] ---> Exchanges cryptographic keys for HTTPS encryption
         |
         | 2. HTTP GET Request (Headers, Cookies, Path)
         v
[ Web Server (Apache / Nginx) ]
         |
         +--> Fetches static HTML/CSS/Images from file storage
         +--> Or invokes Server-Side Engine (PHP) & queries MySQL Database
         |
         | 3. HTTP Response (Status 200 OK, Response Headers, HTML Body)
         v
[ Web Browser Rendering Engine ]
         |
         +--> Parses HTML to build DOM Tree
         +--> Parses CSS to build CSSOM Tree
         +--> Combines into Render Tree -> Computes Layout -> Paints Pixels on Screen
```

3. Detailed Step-by-Step Working Mechanism

1. **URL Input and Parsing:** The user types a URL (e.g., `https://www.example.com/index.html`) into the browser address bar. The browser parses the URL to determine the protocol (`https`), hostname (`www.example.com`), and resource path (`/index.html`).
2. **DNS Resolution:** The browser checks its local cache and the operating system cache. If unlisted, it sends recursive queries to the ISP DNS resolver, checking Root, TLD (.com), and Authoritative nameservers to resolve the hostname into an IP address (e.g., `93.184.216.34`).
3. **Establishing TCP/IP Connection:** The browser initiates a Transmission Control Protocol (TCP) 3-way handshake (SYN, SYN-ACK, ACK) with the server at port 443. For secure HTTPS, a TLS handshake establishes symmetric cryptographic session keys.
4. **Sending HTTP Request:** The browser sends an HTTP `GET` request packet containing request lines, user-agent details, accept headers, and authentication cookies.

5. **Server Processing:** The web server (e.g., Apache, Nginx) receives the request, verifies security permissions, and either retrieves the static document from disk or invokes a script interpreter (e.g., PHP) to dynamically query a database.
6. **Returning HTTP Response:** The server transmits an HTTP response packet consisting of an HTTP status code (e.g., `200 OK`, `404 Not Found`), response headers (Content-Type, Content-Length), and the payload entity body.
7. **Browser Rendering Pipeline:** The browser's layout engine parses the HTML to construct the Document Object Model (DOM), parses CSS to construct the CSS Object Model (CSSOM), merges both into a Render Tree, calculates box geometry (Layout), and rasterizes visual elements to the screen (Painting).

4. Core Web Components Summary Table

Component	Primary Protocol / Standard	Core Functional Role
Client (Browser)	HTTP, HTML5, CSS3, ECMAScript	Provides user interface, initiates requests, and renders web resources.
DNS Server	DNS Protocol (UDP Port 53)	Translates human-readable domain names into numerical IP addresses.
Web Server	HTTP / HTTPS (TCP Port 80 / 443)	Processes requests, executes server scripts, and serves digital content.
Database Server	SQL (e.g. MySQL Port 3306)	Stores, manages, and queries persistent application business data.

5. Concrete Walkthrough Example

When a user accesses `https://www.bcu.ac.in/admissions.php`, the browser queries DNS to locate the IP of `bcu.ac.in`. Upon establishing an encrypted HTTPS connection, the web server executes `admissions.php`, fetches course fees from a MySQL database, bundles the records into an HTML page, and streams it back with status `HTTP/1.1 200 OK`. The browser then displays the rendered admissions table to the user.

Q18. Explain HTML tables and the basic elements used for creating tables.

1. Core Concept of HTML Tables

HTML tables are used to display structured, multi-dimensional tabular data arranged in rows and columns. Tables allow users to view and compare complex datasets such as academic timetables, financial ledgers, product specifications, and student mark sheets. In modern web development, tables are strictly reserved for tabular data and are never used for webpage structural layouts.

2. Fundamental Table Elements and Attributes

- `<table>`: The top-level wrapper container that defines an HTML table.
- `<caption>`: Placed immediately after `<table>` to provide a descriptive title or summary for accessibility.
- `<thead>`: Groups the header rows containing column titles; printed on each page during document printing.
- `<tbody>`: Groups the primary body data rows containing record values.
- `<tfoot>`: Groups footer rows containing column totals, calculations, or footnotes.
- `<tr>`: Defines an individual horizontal table row.
- `<th>`: Defines a table header cell. Rendered in bold, centered text by default.
- `<td>`: Defines a standard table data cell. Rendered regular and left-aligned by default.
- **rowspan:** Attribute of `<th>` or `<td>` that specifies the number of vertical rows a cell spans downward.
- **colspan:** Attribute of `<th>` or `<td>` that specifies the number of horizontal columns a cell spans across.

3. Complete Working HTML Code Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Student Semester Grade Report</title>
  <style>
    table.grade-sheet {
      width: 100%;
      border-collapse: collapse;
      font-family: Arial, sans-serif;
      margin: 15px 0;
    }
    table.grade-sheet caption {
      font-size: 16px;
      font-weight: bold;
      margin-bottom: 8px;
      text-align: center;
    }
    table.grade-sheet th, table.grade-sheet td {
      border: 1px solid #475569;
      padding: 9px 12px;
      text-align: center;
    }
    table.grade-sheet thead th {
      background-color: #1e293b;
      color: #ffffff;
    }
    table.grade-sheet tbody tr:nth-child(even) {
      background-color: #f1f5f9;
    }
    table.grade-sheet tfoot td {
      font-weight: bold;
      background-color: #e2e8f0;
    }
  </style>
</head>
<body>

  <table class="grade-sheet">
    <caption>BCA 5th Semester Examination Marksheet</caption>
    <thead>
      <tr>
        <th rowspan="2">Roll No</th>
        <th rowspan="2">Student Name</th>
        <th colspan="2">Subjects</th>
        <th rowspan="2">Total Marks</th>
        <th rowspan="2">Result</th>
      </tr>
      <tr>
        <th>Data Analytics</th>
        <th>Web Programming</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>101</td>
        <td>Arun Kumar</td>
        <td>85</td>
        <td>90</td>
        <td>175</td>
```

```
        <td>Pass</td>
      </tr>
      <tr>
        <td>102</td>
        <td>Divya Sharma</td>
        <td>78</td>
        <td>88</td>
        <td>166</td>
        <td>Pass</td>
      </tr>
      <tr>
        <td>103</td>
        <td>Naveen Raj</td>
        <td>92</td>
        <td>95</td>
        <td>187</td>
        <td>Pass</td>
      </tr>
    </tbody>
    <tfoot>
      <tr>
        <td colspan="4">Class Average Score</td>
        <td colspan="2">176.0 Marks</td>
      </tr>
    </tfoot>
  </table>

</body>
</html>
```

4. Table Styling and Behavior Summary Table

Element / Attribute	Type	Styling & Structural Purpose
`border-collapse: collapse;`	CSS Property	Merges adjacent cell borders into a single clean rule border.
`rowspan="2"`	HTML Attribute	Stretches a cell vertically across two successive table rows.
`colspan="2"`	HTML Attribute	Stretches a cell horizontally across two adjacent table columns.
`:nth-child(even)`	CSS Selector	Applies alternating row background colors (zebra striping) for readability.

5. Rendered Output Description

The table displays a centered caption at the top. The header shows "Roll No" and "Student Name" spanning vertically across two rows, while "Subjects" spans horizontally across two sub-headers. Data rows feature alternating slate-gray striping. The footer row spans 4 columns horizontally on the left to summarize the class average.

Q19. Explain the basic syntax rules, comments, and code blocks used in PHP.

1. Core Concept of PHP Syntax

PHP (Hypertext Preprocessor) is an open-source, server-side, interpreted scripting language designed primarily for web development. PHP code files are saved with a `.php` extension. When requested, the web server executes the PHP engine, processes the PHP scripts, and returns plain HTML to the client browser.

2. Fundamental Syntax Rules

- **PHP Tags:** PHP script blocks begin with the opening tag `<?php` and end with the closing tag `?>`. In files containing pure PHP code, omitting the closing tag is recommended to avoid accidental whitespace issues.
- **Statement Termination:** Every individual PHP statement must terminate with a semicolon (`;`). Omitting the semicolon results in a parse error.

- **Case Sensitivity Rules:**
  - **Case-Sensitive:** Variable names are strictly case-sensitive. `$score`, `$Score`, and `$SCORE` represent three distinct variables.
  - **Case-Insensitive:** Keywords (e.g., `if`, `else`, `while`, `echo`), built-in functions (e.g., `count()`, `strlen()`), and user-defined functions are case-insensitive.
- **Variable Syntax:** All variables in PHP begin with a dollar sign (`$`), followed by a letter or underscore (e.g., `$userName`, `$_count`). Variables are loosely typed.

3. Comments in PHP

Comments provide documentation for developers and are ignored by the PHP parsing engine:

- **Single-Line Comments:** Denoted by double forward slashes (`//`) or a hash symbol (`#`). Everything following on that line is ignored.
- **Multi-Line Comments:** Denoted by `/*` and closed with `*/`. Used for documenting complex algorithms, functions, or block explanations spanning multiple lines.

4. Code Blocks in PHP

A **code block** is a logical grouping of one or more PHP statements enclosed within curly braces `{ ... }`.
Code blocks define the execution bodies of:

- Conditional decision structures (e.g., `if`, `else`, `switch`).
- Looping iteration structures (e.g., `for`, `while`, `foreach`).
- User-defined function definitions and object-oriented class bodies.

5. Complete Illustrative PHP Script

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>PHP Syntax, Comments, and Code Blocks</title>
</head>
<body>
  <h2>PHP Syntax Demonstration</h2>

  <?php
  // ========================================
  // 1. SINGLE-LINE COMMENTS
  // ========================================
  // This is a C++ style single-line comment
  # This is a Unix shell style single-line comment

  /*
   * 2. MULTI-LINE COMMENT
   * The following variables demonstrate
   * case sensitivity and loose data typing in PHP.
   */
  $courseName = "BCA Web Programming";
  $studentCount = 45;
  $isElective = false;

  // 3. CASE SENSITIVITY DEMONSTRATION
  // Keywords are case-insensitive:
  ECHO "<p>Course: " . $courseName . "</p>";
  echo "<p>Enrolled Students: " . $studentCount . "</p>";

  // 4. CODE BLOCK DEMONSTRATION
  // Using curly braces to encapsulate multi-statement conditional block
  if ($studentCount >= 40) {
      $batchStatus = "Full Batch";
      $roomAllocation = "Auditorium Hall 2";
      echo "<p><strong>Status:</strong> " . $batchStatus . " assigned to " . $roomAllocation . "</p>";
  } else {
      $batchStatus = "Open Batch";
      echo "<p><strong>Status:</strong> Additional admissions permitted.</p>";
  }

  // 5. EMBEDDED LOOP CODE BLOCK
  echo "<ul>";
  for ($i = 1; $i <= 3; $i++) {
      echo "<li>Module Unit " . $i . " Syllabus Active</li>";
  }
  echo "</ul>";
  ?>

</body>
</html>
```

6. Output & Execution Trace

When requested, the PHP engine parses the code blocks, evaluates the `if` condition (45 ≥ 40), assigns the batch variables, and generates standard HTML. The client browser displays the course title, student count, batch status in Auditorium Hall 2, and an unordered list of three module units.

> **Q20. Explain how to retrieve MySQL query results in PHP using mysqli_fetch_array(), mysqli_fetch_assoc(), and mysqli_fetch_row().**

1. Core Concept of Query Result Fetching

When a PHP script executes a SQL `SELECT` query using `mysqli_query()`, the database server returns a result set object representing cursor rows in server memory. To read the retrieved records, PHP provides three specialized fetch functions within the MySQLi extension. Each function fetches a single row from the result set, advances the internal row cursor, and returns `null` when no more records exist.

2. Detailed Examination of Fetch Functions

- **mysqli_fetch_row($result):**
  - Fetches one result row and returns it as an **indexed array** with numeric zero-based indices (`$row[0]`, `$row[1]`).
  - **Advantage:** Fastest and consumes the least memory overhead.
  - **Disadvantage:** Relies strictly on the numerical order of columns in the SQL `SELECT` clause.
- **mysqli_fetch_assoc($result):**
  - Fetches one result row and returns it as an **associative array** where array keys match the table's column names (`$row['col_name']`).
  - **Advantage:** Code is clean, descriptive, and resilient to changes in column ordering.
  - **Disadvantage:** Slightly slower than numeric indexed retrieval.
- **mysqli_fetch_array($result, $result_type):**
  - Fetches one result row as an associative array, a numeric indexed array, or **both**.
  - **Optional Parameter:** `MYSQLI_ASSOC`, `MYSQLI_NUM`, or `MYSQLI_BOTH` (default).
  - **Advantage:** Highly flexible; allows accessing data via `$row[0]` or `$row['student_id']`.
  - **Disadvantage:** Consumes approximately double the memory in default mode due to storing two entries per field.

3. Comparative Summary Table

Fetch Function	Array Type Returned	Key / Index Access Syntax	Memory Overhead	Recommended Use Case
`mysqli_fetch_row()`	Indexed Array	`$row[0]`, `$row[1]`	Minimal	High-performance queries where column order is fixed.
`mysqli_fetch_assoc()`	Associative Array	`$row['name']`, `$row['email']`	Standard	Best standard practice for readable, maintainable web applications.
`mysqli_fetch_array()`	Both Numeric & Assoc (default)	`$row[0]` OR `$row['name']`	Higher (2x keys)	Legacy applications or when dynamic index switching is needed.

4. Complete Working PHP and MySQL Script

```php
<?php
// Database connection credentials
$host     = "localhost";
$username = "root";
$password = "";
$database = "college_db";

// 1. Establish Database Connection
$conn = mysqli_connect($host, $username, $password, $database);
if (!$conn) {
    die("Database Connection Failed: " . mysqli_connect_error());
}

// 2. Execute SQL Query
$sql = "SELECT student_id, student_name, course, marks FROM students ORDER BY student_id ASC";
$result = mysqli_query($conn, $sql);

if ($result && mysqli_num_rows($result) > 0) {
    echo "<h3>Student Record Table (Using mysqli_fetch_assoc)</h3>";
    echo "<table border='1' cellpadding='8' cellspacing='0' style='border-collapse:collapse;'>";
    echo "<tr bgcolor='#f1f5f9'><th>ID</th><th>Name</th><th>Course</th><th>Marks</th></tr>";

    // 3. Iterate through result set row-by-row
    while ($row = mysqli_fetch_assoc($result)) {
        echo "<tr>";
        echo "<td>" . htmlspecialchars($row['student_id']) . "</td>";
        echo "<td>" . htmlspecialchars($row['student_name']) . "</td>";
        echo "<td>" . htmlspecialchars($row['course']) . "</td>";
        echo "<td>" . htmlspecialchars($row['marks']) . "</td>";
        echo "</tr>";
    }
    echo "</table>";

    // 4. Free Result Set Memory
    mysqli_free_result($result);
} else {
    echo "<p>No student records found.</p>";
}

// 5. Close Connection
mysqli_close($conn);
?>
```

5. Output & Loop Mechanism

The `while($row = mysqli_fetch_assoc($result))` statement evaluates truthy on every iteration where a database row is successfully returned. As each record is fetched, its fields are escaped and inserted into table cells. When the last row is passed, the function returns `null`, causing the `while` loop to terminate safely.

Q21. Explain the important features of modern CSS.

1. Core Concept of Modern CSS

Modern CSS (CSS3 and subsequent living specifications) has evolved from a basic visual styling language into a powerful, modular presentation framework. It incorporates modular design paradigms, responsive viewport intelligence, hardware-accelerated animations, and native algorithmic functions, enabling developers to build fluid, responsive web applications without relying on third-party JavaScript layout libraries.

2. Important Features of Modern CSS

- **CSS Custom Properties (Variables):** Defined using the double-hyphen prefix (e.g., `--primary-color: #2563eb;`) and accessed using `var(--primary-color)`. They support cascading inheritance and enable effortless runtime theme toggling (e.g., Dark Mode).
- **Advanced Layout Engines (Flexbox & CSS Grid):**
  - **Flexbox:** One-dimensional layout module for distributing space along rows or columns dynamically.
  - **CSS Grid:** Two-dimensional layout system that organizes content into structured columns and rows simultaneously.
- **Media Queries and Responsive Design:** Uses rules like `@media (max-width: 768px)` and `@media (prefers-color-scheme: dark)` to adapt interface typography, column arrangements, and color schemes to diverse device viewports.
- **Transitions and Keyframe Animations:** Hardware-accelerated animations that create smooth interactive state changes without JavaScript, utilizing `transition: all 0.3s ease;` and `@keyframes` rules.
- **Mathematical Calculation Functions:** Functions such as `calc()`, `min()`, `max()`, and `clamp()` allow dynamic, fluid typography and layout sizing (e.g., `font-size: clamp(1rem, 2.5vw, 2rem);`).
- **Modern Pseudo-Classes & Selectors:** Selectors such as `:is()`, `:where()`, `:has()` (parent selector), and `:not()` reduce stylesheet specificity conflicts and streamline selector lists.

3. Features Summary Table

Modern CSS Feature	Primary Syntax / Mechanism	Impact on Web Development
Custom Properties	`--brand-color: #3b82f6; var(--brand-color)`	Enables centralized theme design tokens and dynamic runtime styling.
CSS Grid Layout	`display: grid; grid-template-columns: repeat(3, 1fr);`	Replaces fragile floats and table layouts with robust two-dimensional grids.
Fluid Math Functions	`width: calc(100% - 40px); font-size: clamp(...)`	Enables scalable responsive dimensions without complex media query breakpoints.
CSS Transitions	`transition: transform 0.25s ease, box-shadow 0.25s;`	Delivers smooth, 60fps GPU-accelerated interactive micro-animations.

4. Complete Modern CSS Demonstration Stylesheet

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Modern CSS Feature Demonstration</title>
  <style>
    /* 1. CSS CUSTOM PROPERTIES (VARIABLES) */
    :root {
      --primary: #2563eb;
      --surface: #ffffff;
      --text: #0f172a;
      --shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1);
      --radius: 8px;
    }

    /* System Dark Mode Detection */
    @media (prefers-color-scheme: dark) {
      :root {
        --primary: #60a5fa;
        --surface: #1e293b;
        --text: #f8fafc;
        --shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.4);
      }
    }

    body {
      background-color: var(--surface);
      color: var(--text);
      font-family: system-ui, sans-serif;
      padding: 20px;
    }

    /* 2. FLUID TYPOGRAPHY USING CLAMP */
    h1 {
      font-size: clamp(1.5rem, 4vw, 2.5rem);
      color: var(--primary);
      text-align: center;
    }

    /* 3. CSS GRID CONTAINER */
    .card-grid {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
      gap: 20px;
      max-width: 900px;
      margin: 30px auto;
    }

    /* 4. TRANSITIONS AND HOVER EFFECTS */
    .feature-card {
      background-color: var(--surface);
      border: 1px solid #cbd5e1;
      border-radius: var(--radius);
      padding: 20px;
      box-shadow: var(--shadow);
      transition: transform 0.3s ease, box-shadow 0.3s ease;
    }

    .feature-card:hover {
      transform: translateY(-5px);
```

```
      box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.2);
    }
  </style>
</head>
<body>
  <h1>Modern Responsive Web Components</h1>
  <div class="card-grid">
    <div class="feature-card">
      <h3>CSS Grid</h3>
      <p>Two-dimensional responsive track layouts.</p>
    </div>
    <div class="feature-card">
      <h3>CSS Variables</h3>
      <p>Dynamic variables supporting instant dark mode.</p>
    </div>
    <div class="feature-card">
      <h3>Smooth Transitions</h3>
      <p>Micro-animations with zero JavaScript overhead.</p>
    </div>
  </div>
</body>
</html>
```

5. Rendered Output Description

The layout displays three modern cards arranged in a responsive grid. The typography scales fluidly across device widths via `clamp()`. When hovering over any card, a GPU-accelerated transition lifts the card upward by 5px and deepens its shadow. If the operating system enables dark mode, all color tokens adapt immediately without reloading.

Q22. Explain the if, if...else, and switch statements in JavaScript with examples.

1. Core Concept of Conditional Control Flow

Conditional statements allow JavaScript programs to evaluate dynamic expressions and execute specific branches of code based on whether a condition evaluates to `true` or `false`. They form the foundational logic for user authentication, form validation, error handling, and business workflows.

2. Detailed Examination of Decision Structures

- **The `if` Statement:**
  - Evaluates a condition in parentheses. If truthy, the code block inside curly braces executes; otherwise, it is skipped.
  - **Syntax:** `if (condition) { /* statements */ }`
- **The `if...else` Statement:**
  - Provides an alternative fallback path. If the condition is true, the `if` block executes; if false, the `else` block executes.
  - **Syntax:** `if (condition) { /* true code */ } else { /* false code */ }`
- **The `if...else if...else` Ladder:**
  - Tests multiple sequential conditions in hierarchical order. Once a condition is met, its corresponding block executes and the rest of the ladder is skipped.
- **The `switch` Statement:**
  - A multi-branch statement that evaluates an expression against multiple discrete `case` clauses using **strict equality** (`===`).
  - **break:** Terminates the switch block; omitting `break` causes execution to "fall through" into subsequent cases.
  - **default:** Optional clause that executes if no matching case is satisfied.

3. Comparison: if...else vs. switch

Dimension	if...else Statement	switch Statement
Expression Type	Evaluates boolean expressions, relational comparisons, or logical ranges.	Evaluates single scalar values (integer, character, string, boolean).
Comparison Operator	Supports all comparison operators (`>` , `<=` , `!=` , `&&` , `||`).	Strictly uses strict equality comparison (`===`).
Execution Flow	Evaluates conditions sequentially from top to bottom.	Jumps directly to the matched case label using internal jump tables.
Best Suited For	Range checking (e.g., test scores between 80 and 100).	Fixed discrete options (e.g., menu choices, day numbers, status codes).

4. Complete Working JavaScript Demonstration Program

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Conditional Statements</title>
</head>
<body>
  <h2>JavaScript Conditional Flow Demonstrations</h2>
  <div id="output" style="font-family:sans-serif; line-height:1.8;"></div>

  <script>
    var outputDiv = document.getElementById("output");
    var htmlContent = "";

    // ========================================
    // 1. IF...ELSE IF...ELSE LADDER (Grade Evaluator)
    // ========================================
    var marks = 85;
    var grade = "";

    if (marks >= 90) {
      grade = "A+ (Outstanding)";
    } else if (marks >= 75) {
      grade = "A (Distinction)";
    } else if (marks >= 60) {
      grade = "B (First Class)";
    } else if (marks >= 40) {
      grade = "C (Pass)";
    } else {
      grade = "F (Fail)";
    }

    htmlContent += "<p><strong>Score:</strong> " + marks + " marks &rarr; <strong>Grade:</strong> " + grade +

    // ========================================
    // 2. SWITCH STATEMENT (Day of Week Evaluator)
    // ========================================
    var dayNumber = 3;
    var dayName = "";

    switch (dayNumber) {
      case 1:
        dayName = "Monday";
        break;
      case 2:
        dayName = "Tuesday";
        break;
      case 3:
        dayName = "Wednesday";
        break;
      case 4:
        dayName = "Thursday";
        break;
      case 5:
        dayName = "Friday";
        break;
      case 6:
        dayName = "Saturday";
        break;
      case 7:
```

```
        dayName = "Sunday";
        break;
      default:
        dayName = "Invalid Day Number";
    }

    htmlContent += "<p><strong>Day Code:</strong> " + dayNumber + " &rarr; <strong>Day:</strong> " + dayName +

    outputDiv.innerHTML = htmlContent;
  </script>
</body>
</html>
```

5. Output & Execution Trace

For `marks = 85`, the first condition (85 ≥ 90) evaluates to false, but the second condition (85 ≥ 75) evaluates to true; the script sets the grade to "A (Distinction)" and exits the ladder. For `dayNumber = 3`, the switch matches `case 3:` strictly, assigns "Wednesday", and encounters `break` to terminate the switch block.

Q23. (a) Explain the role of AJAX in web programming with a suitable example. [4 Marks] (b) Explain the alert(), confirm(), and prompt() popup boxes with examples. [4 Marks]

(a) Explain the role of AJAX in web programming with a suitable example. [4 Marks]

1. Core Concept & Expansion: AJAX stands for **Asynchronous JavaScript and XML**. It is not an independent programming language, but a client-side web development technique that allows web applications to communicate asynchronously with a web server in the background without refreshing the entire web page.

2. Key Roles and Benefits:

- **Partial Page Updates:** Dynamically updates specific DOM container elements without tearing down the existing browser state or causing page flickers.
- **Bandwidth & Resource Efficiency:** Transmits and receives only raw data payloads (predominantly lightweight JSON), avoiding redundant re-transmission of layout HTML, CSS, and scripts.
- **Enhanced User Experience:** Delivers continuous, desktop-like responsiveness for real-time validation, infinite scrolling, live notifications, and search autocomplete.

3. Architectural Lifecycle Diagram:

```
[User Event (keyup)] ---> [JS creates XMLHttpRequest / fetch()] ---> Sends HTTP Request
                                                                          |
[DOM Updated Silently] <--- [JS receives JSON Data] <--- [Server processes & responds]
```

4. Concrete Code Example (Live Search / Autocomplete):

```
// Asynchronous fetch function triggered on input
function searchCollegeCourses(keyword) {
  fetch("search.php?query=" + encodeURIComponent(keyword))
    .then(response => response.json())
    .then(data => {
      let resultBox = document.getElementById("results");
      resultBox.innerHTML = "";
      data.forEach(item => {
        resultBox.innerHTML += "<div>" + item.courseName + "</div>";
      });
    })
    .catch(error => console.error("AJAX Error:", error));
}
```

(b) Explain the alert(), confirm(), and prompt() popup boxes with examples. [4 Marks]

1. Core Concept: In JavaScript, popup boxes are built-in modal dialog windows provided by the browser's `window` object. Because they are modal, they temporarily pause JavaScript script execution and prevent user interaction with the underlying page until dismissed.

2. Detailed Explanation of the Three Popup Boxes:

- **alert(message):**
  - Displays an informational notification message with a single "OK" button.
  - **Return Value:** Always returns `undefined`.
  - **Syntax:** `alert("Data saved successfully!");`
- **confirm(message):**
  - Displays a verification dialog with "OK" and "Cancel" buttons.
  - **Return Value:** Returns `true` if user clicks OK; returns `false` if user clicks Cancel.
  - **Syntax:** `var answer = confirm("Do you wish to delete this record?");`
- **prompt(message, defaultValue):**
  - Displays an input dialog with a text box, an "OK" button, and a "Cancel" button, prompting the user for text input.
  - **Return Value:** Returns a `string` containing the user's input if OK is clicked; returns `null` if Cancel is clicked.
  - **Syntax:** `var name = prompt("Enter your username:", "Student");`

3. Complete Interactive JavaScript Program:

```
<script>
  // 1. Alert Box
  alert("Welcome to the Student Examination Portal!");

  // 2. Prompt Box
  var studentName = prompt("Please enter your name:", "Guest");

  if (studentName !== null && studentName.trim() !== "") {
    // 3. Confirm Box
    var proceed = confirm("Hello, " + studentName + "! Do you wish to start the exam?");
    if (proceed) {
      alert("Exam started. You have 3 hours.");
    } else {
      alert("Exam session cancelled.");
    }
  } else {
    alert("Username entry was cancelled.");
  }
</script>
```
