

1. i. What will be the output of the following PHP code?

```
1 <?php
2 $i=0;
3 for(++$i; ++$i; ++$i)
4 {
5     print $i;
6     if ($i == 4)
7         break;
8 }
9 ?>
```

Ans: 24(its 2 and 4)

main.php	Run	Output
<pre>1 <?php 2 3 \$i=0; 4 for(++\$i; ++\$i; ++\$i){ 5 print \$i; 6 if (\$i == 4) 7 break; 8 } 9 10 ?></pre>		24 === Code E

- ii. What will be the output of the following PHP code?

```
1. <?php
2. $names = array("Sam", "Bob", "Jack");
3. echo $names[0]."is the brother of ".$names[1]." and ".$names[1]." ".$names[2]." brother;
4. ?>
```

Ans: Here, since the variable \$brother is not defined, php will throw an undefined error. but still print the above.

Output:

Warning: Undefined variable \$brother in **D:\xampp\htdocs\final\index.php** on line 4
Samis the brother of Boband Bob.

main.php	Output
<pre>1 <?php 2 \$a = "clue"; 3 \$a .= "get"; 4 echo "\$a"; 5 ?></pre>	<pre>clueget === Code Ex</pre>

<pre>1 <?php 2 \$team = "arsenal"; 3 switch (\$team) { 4 case "manu": 5 echo "I love man u"; 6 case "arsenal": 7 echo "I love arsenal"; 8 case "manc": 9 echo "I love manc"; 10 } 11 ?></pre>	<pre>I love arsenalI love manc === Code Execution Successful ===</pre>
---	---

```
// Explanation:
// The variable $team is assigned the value "arsenal".
// The switch statement checks the value of $team.
// The case "manu" is checked first. Since $team is "arsenal" and
// not "manu", this case is skipped.
// The case "arsenal" is checked next. Since $team is "arsenal",
// this case is matched.
// "I love arsenal" is printed.
// Because there is no 'break' statement after the "arsenal" case,
// the execution "falls through" to the next case ("manc").
// The case "manc" is checked.
// "I love manc" is printed.
// Since there are no more cases, the switch statement ends.
```

statement

Find the output of the following PHP code?

```
<?php
$num = "1";
$num1 = "2";
print $num+$num1;
?>
```

```
1 <?php
2 $num="1";
3 $num1="2";
4 print $num+$num1;
5 ?>
```

```
3
=== Code Execution Successful ===
```

Ans: output is 3.

When using `+` with strings that contain numeric values, PHP automatically converts them into integers or floats.

4. a) What are JavaScript Objects? Give some examples of JavaScript built-in objects. 3

b) Write a JavaScript program to evaluate the following mathematical expression: 5

$$1 + \frac{2}{2!} + \frac{3}{3!} + \dots + \frac{n}{n!}$$

c) Create a regex for JavaScript which can validate any email id. 2

Ans:

Here are the answers to the given questions:

(a) What are JavaScript Objects? Give some examples of JavaScript built-in objects.

JavaScript Objects are collections of key-value pairs, where keys are strings (or Symbols) and values can be any data type. Objects allow us to store structured data and represent real-world entities.

Example:

```
const person = {
  firstName: "John",
```

```

lastName : "Doe",

id        : 5566,

fullName : function() {

    return this.firstName + " " + this.lastName;

}

};

```

In JavaScript, almost "everything" is an object.

- Objects are objects
- Maths are objects
- Functions are objects
- Dates are objects
- Arrays are objects
- Maps are objects
- Sets are objects

All JavaScript values, except primitives, are objects.

Examples of JavaScript built-in objects:

1. **Math** – Provides mathematical constants and functions (e.g., `Math.PI`, `Math.sqrt()`).
2. **Date** – Used to handle date and time (e.g., `new Date()`).
3. **String** – Allows manipulation of string values (e.g., `"Hello".toUpperCase()`).
4. **Array** – Represents ordered collections of data (e.g., `[1, 2, 3].length`).
5. **RegExp** – Used for pattern matching with regular expressions (e.g., `/abc/.test("abcdef")`).
6. **JSON** – Provides methods to parse and stringify JSON (e.g., `JSON.parse()`).
7. **Object** – The base object for all JavaScript objects (e.g., `Object.keys({a:1, b:2})`).

(b) JavaScript Program to Evaluate the Given Mathematical Expression

The given mathematical expression is:

$$1 + 22! + 33! + \dots + nn! + \frac{2}{2!} + \frac{3}{3!} + \dots + \frac{n}{n!}$$

Here's the JavaScript program to compute this sum:

```
function factorial(n) {
  if (n === 0 || n === 1) return 1;
  return n * factorial(n - 1);
}

function evaluateExpression(n) {
  let sum = 1; // Start with 1
  for (let i = 2; i <= n; i++) {
    sum += i / factorial(i);
  }
  return sum;
}

let n = 5; // You can change the value of n
console.log("Result:", evaluateExpression(n));
```

Explanation:

- The `factorial()` function calculates the factorial of a number recursively.
- The `evaluateExpression(n)` function loops through the series and calculates each term.
- The result is printed to the console.

(c) Create a Regex for JavaScript That Can Validate Any Email ID

Here's a **regular expression** to validate an email in JavaScript:

```
function validateEmail(email) {
  const regex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
  return regex.test(email);
}

// Example Usage:
console.log(validateEmail("example@mail.com")); // true
console.log(validateEmail("invalid-email")); // false
```

Explanation:

- `^[a-zA-Z0-9._%+-]+` → Starts with alphanumeric characters and common symbols.
- `@` → Must have an "@" symbol.
- `[a-zA-Z0-9.-]+` → Domain name (e.g., `mail.com`).
- `\.[a-zA-Z]{2,}$` → Ends with a dot followed by at least two letters (e.g., `.com`, `.org`).

Easier version:

Example of a Simpler Email Regex

A **basic** but effective email validation regex:

```
const emailRegex = /^[\w.-]+@[a-zA-Z\d.-]+\.[a-zA-Z]{2,}$/;
```

- Uses `\w` (equivalent to `[a-zA-Z0-9_]`) instead of `[a-zA-Z0-9._%+-]`.
- Keeps domain and extension flexible.

Sessions vs Cookies in Web Development

Both **sessions** and **cookies** are used to store user data across multiple pages of a website. However, they differ in where they store data, how they work, and their security implications.

1. What is a Cookie?

A **cookie** is a small piece of **data stored in the user's browser**. Websites use cookies to remember user preferences, login information, and other details across different visits.

How Cookies Work:

1. When a user visits a website, the server sends a cookie to the user's browser.
2. The browser stores the cookie.
3. On subsequent visits, the browser automatically sends the cookie back to the server.
4. The server reads the cookie and personalizes the experience.

Example: Setting and Retrieving Cookies in JavaScript

```
// Set a cookie (expires in 7 days)
```

```
document.cookie = "username=John; expires=" + new Date(2025, 1, 1).toUTCString();
```

```
// Retrieve a cookie
```

```
console.log(document.cookie); // Output: username=John
```

Types of Cookies

1. **Session Cookies** – Deleted when the browser is closed.
 2. **Persistent Cookies** – Stored for a specific duration (e.g., days or months).
 3. **Secure Cookies** – Only sent over HTTPS connections.
 4. **HttpOnly Cookies** – Cannot be accessed via JavaScript (only server-side).
-

2. What is a Session?

A **session** is a temporary **storage mechanism on the server** that stores user-specific data while they interact with a website.

How Sessions Work:

1. A user visits a website, and the server creates a **unique session ID**.
2. The session ID is sent to the browser (usually as a cookie).
3. The browser sends this session ID with each request.
4. The server uses this session ID to retrieve the stored session data.

Example: Creating and Using Sessions in PHP

```
// Start a session
```

```
session_start();
```

```
// Store session data
```

```
$_SESSION["username"] = "John";
```

```
// Retrieve session data
```

```
echo "Welcome, " . $_SESSION["username"]; // Output: Welcome, John
```

Key Characteristics of Sessions

- Stored on the **server** instead of the client.
 - Automatically expires after inactivity.
 - More **secure** than cookies since data is not exposed to the user.
-

3. Differences Between Sessions and Cookies

Feature	Cookies	Sessions
Storage Location	Stored in the browser (client-side)	Stored on the server
Security	Less secure (visible and modifiable by users)	More secure (data stays on the server)
Data Size	Limited to 4KB per cookie	Can store large amounts of data
Expiry	Can be set to expire after a specific time	Expires when the user closes the browser or after inactivity
Performance	Fast, as data is stored locally	Slower, as each request must fetch data from the server
Use Cases	Remembering user preferences, login credentials, tracking	Storing temporary user data (e.g., shopping cart, authentication)

4. When to Use What?

- **Use Cookies** when:
 - You need to store small, non-sensitive data like user preferences.
 - The data should persist even after the session ends (e.g., "Remember Me" login).
 - **Use Sessions** when:
 - You need to store sensitive user information (e.g., authentication, cart items).
 - The data should expire when the user logs out or leaves the site.
-

5. Example: Using Both Together

A common approach is to store session IDs in cookies:

```
// Set a session
```


```
session_start();
```

```
$_SESSION["user_id"] = 123;
```

```
// Set a cookie with session ID
```

```
setcookie("session_id", session_id(), time() + 3600, "/"); // 1-hour expiry
```

2. a. Assume, in a system **login.php**, **index.php**, **logout.php** are the three php pages. Username and password will be asked in **login.php**. **Id**, **username** and **password** are three attributes of a table “**user**” in a database “**userinfo**”. The username and password should be checked for the authentication of any user in the system. If the username is found in the database then you have to redirect to the **index.php** by setting the session variable by user id. Implement the appropriate logout by destroying session variables and associated session data. When any user checks the remember me option, set the cookies expiration time to one day. A simple scenario is given in Figure 1 for more clarification. 5



Username

Password

☐ Remember Me

Log In

Welcome Bob!

[Logout](#)

login.phpindex.php

Figure: 1

Design a proper PHP login-logout scheme with PHP session, cookies and the specification given above.

Step 1: Database Setup

Step 1: Database Setup

We assume a database `userinfo` with a table `user`:

```
sql
CREATE DATABASE userinfo;

USE userinfo;

CREATE TABLE user (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);
```

Copy Edit

Explain

Insert a sample user with password hashing:

```
php
<?php
$conn = new mysqli("localhost", "root", "", "userinfo");
$password = password_hash("123456", PASSWORD_DEFAULT);
$conn->query("INSERT INTO user (username, password) VALUES ('Bob', '$password')");
$conn->close();
?>
```

Copy Edit

Explain

Step 2: login.php (User Authentication)

This script:

- Accepts username and password
- Checks if "Remember Me" is selected and sets cookies

- Uses `password_verify()` for security
- Redirects to `index.php` if login is successful,

```
× Login Script

1  <?php
2  session_start(); // Start a session
3
4  $conn = new mysqli("localhost", "root", "", "userinfo"); // Connect to MySQL database
5
6  if ($_SERVER["REQUEST_METHOD"] == "POST") {
7      $username = $_POST["username"];
8      $password = $_POST["password"];
9      // Prepare an SQL statement to prevent SQL Injection
10     $stmt = $conn->prepare("SELECT id, password FROM user WHERE username = ?"); // Use a prepared statement instead of
11                                             //directly injecting user input into SQL query
12     $stmt->bind_param("s", $username); // Bind the user-provided username as a string parameter to prevent SQL injection
13     $stmt->execute(); // Execute the prepared statement
14     $stmt->store_result(); // Store the result to check if the username exists
15
16     if ($stmt->num_rows > 0) { // If a row is found, it means the username exists in the database
17         $stmt->bind_result($user_id, $hashed_password); // Bind the retrieved columns to PHP variables
18         $stmt->fetch(); // Fetch the result into the bound variables
19
20         // Verify entered password with the hashed password stored in the database
21         if (password_verify($password, $hashed_password)) { // Secure password verification using hash comparison
22             $_SESSION["user_id"] = $user_id; // Set session variable for user ID
23             $_SESSION["username"] = $username; // Set session variable for username
24
25             // If "Remember Me" is checked, set a cookie for 1 day
26             if (isset($_POST["remember_me"])) {
27                 setcookie("username", $username, time() + 86400, "/"); // Create a cookie to remember the username for 1 day
28             }
29             header("Location: index.php"); // Redirect to index.php after successful login
30             exit();
31         } else {
32             echo "Invalid password!"; // Error message if password verification fails
33         }
34     } else {
35         echo "Username not found!"; // Error message if no such username exists
36     }
37     $stmt->close(); // Close the prepared statement to free up resources
38 }
39 ?>
40
```

```

        header("Location: index.php");
        exit();
    } else {
        echo "Invalid username or password.";
    }
}
?>

<form method="post">
    <label>Username:</label>
    <input type="text" name="username" required>
    <br>
    <label>Password:</label>
    <input type="password" name="password" required>
    <br>
    <input type="checkbox" name="remember"> Remember Me
    <br>
    <button type="submit">Log In</button>
</form>

```

Step 3: index.php (User Dashboard)

This script:

- Uses session or cookie to maintain login state
- Displays the username
- Provides a logout option

✕ Welcome Page

```

1  <?php
2  session_start(); // Start the session
3
4  // Retrieve username from session if available, otherwise from cookie
5  $username = $_SESSION['username'] ?? $_COOKIE['username'] ?? "Guest";
6
7  ?>
8
9  <h1>Welcome, <?php echo htmlspecialchars($username); ?>!</h1>
10 <a href="logout.php">Logout</a>
11

```

Step 4: logout.php (Destroy Session and Cookies)

This script:

- Ends the session
- Deletes login cookies
- Redirects to `login.php`

✕ Logout Script

```
1  <?php
2  session_start();
3  session_unset(); // Remove all session variables
4  session_destroy(); // Destroy the session
5
6  // Delete the username cookie by setting expiration time to the past
7  setcookie("username", "", time() - 3600, "/");
8
9  header("Location: login.php"); // Redirect back to login page
10 exit();
11 ?>
12
```

3. a. What do you mean by MVC? Explain the advantages and disadvantages of Code First Approach and DB First Approach.

MVC Architecture (Model-View-Controller)

The **MVC (Model-View-Controller)** architecture is a **design pattern** used in software development, particularly in web applications, to separate concerns and improve maintainability.

Why Use MVC?

- Enhances **code organization**.
- Facilitates **scalability and maintainability**.
- Encourages **separation of concerns** (each component has a specific role).

- Helps in **parallel development** (e.g., frontend and backend can be developed separately).
-

Components of MVC

MVC consists of **three interconnected components**:

1. Model (Data & Business Logic)

- Represents the **data** and **business logic** of the application.
 - Retrieves, processes, and stores data in a **database**.
 - Does **not** directly interact with the user interface.
 - Example: Handling **database queries**, **validations**, and **calculations**.
-

2. View (User Interface)

- Handles **UI and presentation**.
 - Displays **data received from the Model**.
 - **Does not** contain business logic.
 - Example: **HTML**, **CSS**, **JavaScript** files rendering the frontend.
-

3. Controller (Handles User Requests)

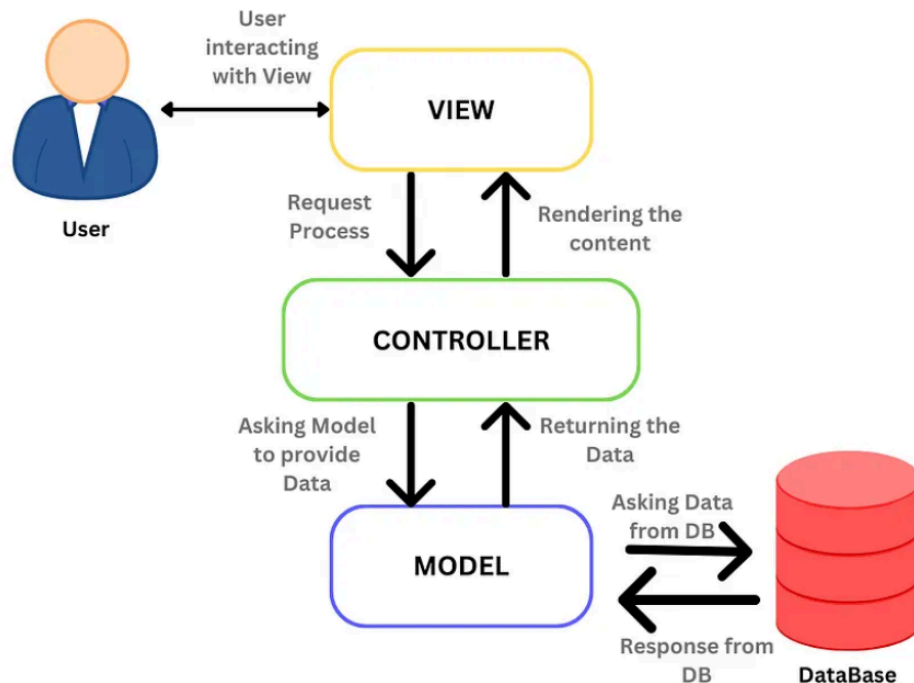
- Acts as an **intermediary** between the Model and View.
 - Handles **user input, requests, and interactions**.
 - Calls the **Model** for data and updates the **View**.
-

How MVC Works Together

1. The **user requests** a page (e.g., `GET /profile/1`).

2. The **Controller** processes the request and fetches data from the **Model**.
3. The **Model** interacts with the database and returns the requested data.
4. The **Controller** sends the data to the **View**.
5. The **View** displays the data as a **web page**.

MVC Flowchart



Advantages of MVC

- **Separation of Concerns** → Easier to manage and debug code.
- **Reusability** → Views and Models can be reused in different parts of the application.
- **Parallel Development** → Teams can work on different components independently.
- **Scalability** → Suitable for large and complex applications.

Real-Life Example of MVC

Example: Logging into a Website

1. **User enters credentials and clicks "Login"** → Controller receives the request.
2. **Controller sends data to the Model** → Model verifies user credentials in the database.

3. **Model returns the authentication result** → Controller decides whether to log in the user.
 4. **Controller loads the View** → View displays a success message or error.
-

MVC in Different Frameworks

- **PHP** → Laravel, CodeIgniter
 - **Python** → Django, Flask
 - **JavaScript** → React (kind of), Angular, Express.js
 - **Java** → Spring MVC
 - **.NET** → ASP.NET MVC
-

Conclusion

- MVC **separates concerns** into **Model (Data)**, **View (UI)**, and **Controller (Logic)**.
- It **enhances maintainability** and **scalability**.
- **Many modern frameworks** are based on MVC.

Code-First Approach vs. Database-First Approach

When working with Object-Relational Mapping (ORM) tools like **Entity Framework (EF)**, developers have two main approaches to defining and interacting with a database: **Code-First** and **Database-First**.

1. Code-First Approach

What is Code-First?

The **Code-First approach** allows developers to define the database structure using C# or another programming language. The database is automatically generated based on the code definitions. Instead of designing the database manually, developers create **classes (models)** that represent tables in the database.

How It Works:

1. The developer writes C# classes to define entities.
2. The ORM tool (like Entity Framework) generates the database schema based on these classes.
3. Database migrations can be used to update the schema as the code evolves.

Advantages of Code-First Approach:

- **Flexibility for Developers:** Developers can design the database using code without relying on database administrators.
- **Version Control:** Since the database structure is defined in code, changes can be tracked using Git or other version control tools.
- **Database Independence:** The same code can be used with different database management systems (DBMS) without modifying SQL scripts.
- **Easier Development Process:** Eliminates the need to manually update the database schema when making changes in the application.

Disadvantages of Code-First Approach:

- **Requires ORM Knowledge:** Developers must understand how migrations and ORM tools work.
 - **Not Ideal for Complex Databases:** If the database has many existing relationships and constraints, defining everything in code can be challenging.
 - **Potential Performance Issues:** Auto-generated queries may not be as optimized as manually written SQL queries.
-

2. Database-First Approach

What is Database-First?

The **Database-First approach** is the traditional method where the database schema is designed manually using SQL before the application code is written. ORM tools then generate the code (models) based on the existing database.

How It Works:

1. The developer or database administrator creates the database schema manually.
2. The ORM tool reads the database structure and generates entity classes based on tables and relationships.
3. The application interacts with the database using these generated models.

Advantages of Database-First Approach:

- **Best for Existing Databases:** Useful when working with legacy databases or pre-existing systems.
- **Better Control Over Schema:** Database administrators can manually optimize queries, indexes, and relationships for better performance.
- **Structured and Optimized:** Since the database is designed first, it ensures proper normalization and indexing.
- **Familiar for Database Teams:** Ideal when working in environments where database administrators manage the schema separately from developers.

Disadvantages of Database-First Approach:

- **Less Flexibility:** Changes in the database require manual updates to the application code.
 - **Harder Version Control:** Keeping track of schema changes is more complex compared to Code-First, where changes are in the codebase.
 - **More Manual Work:** Every change to the database requires updates to the ORM-generated code, making development slower.
-

Which Approach to Choose?

Feature	Code-First Approach	Database-First Approach
Best for	New projects, fast development	Existing databases, complex schema

Flexibility	High – database is generated from code	Low – schema changes require manual updates
Performance	May generate less optimized queries	Manually optimized queries possible
Version Control	Easy – changes tracked in code	Harder – database schema tracking required
Required Knowledge	ORM & migrations	SQL & database design

When to Choose Code-First?

- When starting a **new project**.
- When the development team is **more experienced with programming** than databases.
- When you want **database portability** (e.g., switching from MySQL to PostgreSQL).

When to Choose Database-First?

- When working with an **existing database**.
- When database administrators handle the schema separately from developers.
- When **manual optimization** of database performance is required.

Both approaches have their use cases, and the right choice depends on the project requirements.

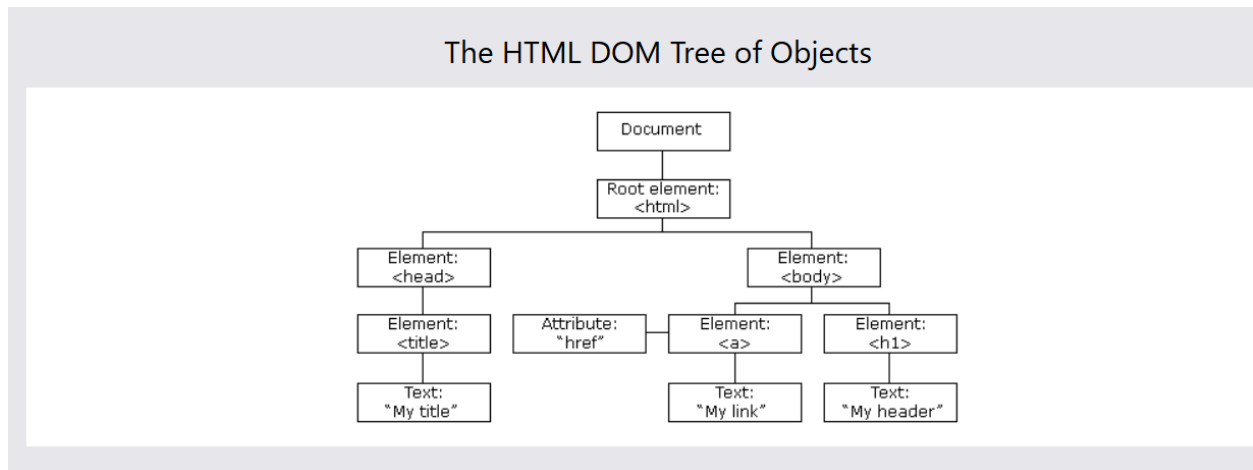
Que. Explain DOM and css selectors with proper example

DOM (Document Object Model) and CSS Selectors Explained with Examples

What is DOM?

The **Document Object Model (DOM)** is a programming interface for web documents. It represents the structure of a webpage as a tree-like hierarchy where each HTML element is a node.

When a web page is loaded, the browser converts the HTML into a DOM structure, allowing JavaScript to manipulate elements dynamically.



Example: Manipulating DOM Using JavaScript

```
html
Copy Edit

<!DOCTYPE html>
<html lang="en">
<head>
  <title>DOM Example</title>
</head>
<body>
  <h1 id="heading">Hello, World!</h1>
  <button onclick="changeText()">Click Me</button>

  <script>
    function changeText() {
      document.getElementById("heading").innerText = "Hello, JavaScript!";
    }
  </script>
</body>
</html>
```

Explanation:

- `document.getElementById("heading")` selects the `<h1>` element.
 - `innerText = "Hello, JavaScript!"` modifies its content dynamically.
-

What are CSS Selectors?

CSS selectors are patterns used to select and style HTML elements. They allow styling elements based on their tag, class, ID, attributes, etc.

Common CSS Selectors

Selector	Description	Example
<code>*</code>	Selects all elements	<code>* { color: red; }</code>
<code>tagname</code>	Selects all elements of that tag	<code>p { font-size: 16px; }</code>
<code>#id</code>	Selects an element with a specific ID	<code>#heading { color: blue; }</code>
<code>.class</code>	Selects all elements with a specific class	<code>.button { background-color: yellow; }</code>
<code>[attribute]</code>	Selects elements with a specific attribute	<code>input[type="text"] { border: 2px solid black; }</code>

Example: Using CSS Selectors

html

CopyEdit

Explain

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    #main-title { color: red; } /* ID Selector */
    .highlight { background-color: yellow; } /* Class Selector */
    p { font-size: 18px; } /* Tag Selector */
  </style>
</head>
<body>
  <h1 id="main-title">This is a Heading</h1>
  <p class="highlight">This is a paragraph with a highlighted background.</p>
</body>
</html>
```

Explanation:

- #main-title applies styles to the <h1> with that ID.
- .highlight applies a yellow background to elements within that class.
- p applies styles to all <p> elements.

Key Differences Between DOM and CSS Selectors

Feature	DOM	CSS Selectors
Purpose	Manipulates elements dynamically	Styles elements visually
Uses	JavaScript	CSS

Example	<code>document.getElementById("demo")</code>	<code>#demo { color: red; }</code>
---------	---	------------------------------------

3. a)	What is the difference between a block-level and an inline-level element in HTML?	2
b)	Write an HTML code snippet that creates a form with a mandatory text input field and a submit button.	3
c)	Explain how <canvas> element works and provide a simple example of its usage.	3
d)	What is the <iframe> element? How it is different from <embed> and <object>?	2

Answers to the Given Questions

(a) Difference Between Block-Level and Inline-Level Elements in HTML

Feature	Block-Level Elements	Inline-Level Elements
Behavior	Start on a new line and take up full width	Stay within the same line as surrounding elements
Examples	<code><div></code> , <code><p></code> , <code><h1></code> - <code><h6></code> , <code><section></code>	<code></code> , <code><a></code> , <code></code> , <code></code>
Use Cases	Used for structuring the layout	Used for styling parts of text or grouping within block elements

Example:

html

CopyEdit

```
<p>This is a <span style="color: blue;">blue inline text</span> inside a paragraph.</p>
<div>This is a block element.</div>
```


(b) HTML Code Snippet for a Form with a Mandatory Text Input Field and Submit Button

```
html Copy Edit  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <title>Form Example</title>  
</head>  
<body>  
  <form action="submit.php" method="POST">  
    <label for="name">Name:</label>  
    <input type="text" id="name" name="name" required> <!-- Required field -->  
    <button type="submit">Submit</button>  
  </form>  
</body>  
</html>
```

Explanation:

- `<form>` creates a form with `action="submit.php"` for submission.
- `<input>` is a text field with `required`, making it mandatory.
- `<button>` submits the form.

(c) `<canvas>` Element Explanation and Example

The `<canvas>` element is used for drawing graphics dynamically via JavaScript. It can render shapes, images, and animations.

Example: Drawing a Circle

Example: Drawing a Circle

html

Copy Edit

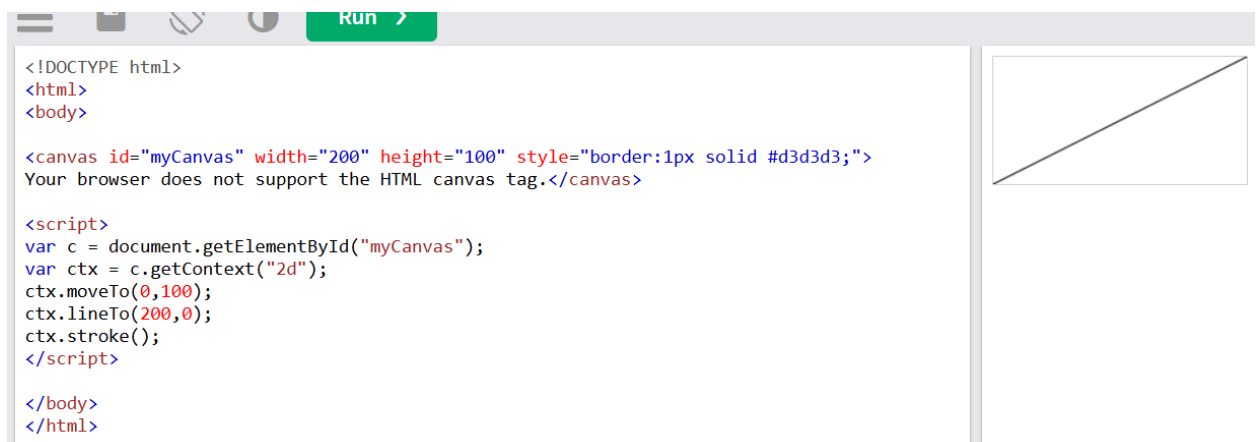
```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Canvas Example</title>
</head>
<body>
  <canvas id="myCanvas" width="200" height="200" style="border:1px solid black;"></canvas>

  <script>
    var canvas = document.getElementById("myCanvas");
    var ctx = canvas.getContext("2d");
    ctx.beginPath();
    ctx.arc(100, 100, 50, 0, 2 * Math.PI);
    ctx.fillStyle = "blue";
    ctx.fill();
  </script>
</body>
</html>
```

Explain

Explanation:

- `getContext("2d")` gets a 2D drawing space.
- `arc(x, y, radius, startAngle, endAngle)` draws a circle.
- `fillStyle = "blue"` fills it with color.





(d) **<iframe>** Element and Its Differences from **<embed>** and **<object>**

- **<iframe>** (Inline Frame): Used to embed another HTML page inside the current one.
- **<embed>**: Used to embed external content like videos, PDFs, or flash.
- **<object>**: More versatile, supports embedding multiple types of media with parameters.



Key Differences

Element	Purpose	Common Use
<iframe>	Embeds another webpage	Embedding YouTube videos, Google Maps

<code><embed></code>	Embeds external media	Audio, video, Flash content
<code><object></code>	Embeds media with extra attributes	PDFs, Java applets

5. a)	Distinguish between "padding" and "margin" area of CSS box model.	2
b)	What are the differences between GET and POST methods in form submitting? Give the case where we can use GET and where we can use POST methods.	3
c)	What does HTML stand for? Write the image tag required to link from the page <i>links.html</i> to the image <i>contact.jpg</i> when the files are organized as follows:	3
	<ul style="list-style-type: none"> i. both in the same folder ii. the <i>contact.jpg</i> is located in a subfolder called "<i>images</i>" iii. the <i>links.html</i> page is located in a subfolder called "<i>otherfiles</i>" 	
d)	Write the differences between "id" and "class" attribute of an HTML element.	2

`<object width="200" height="200" data="https://www.youtube.com/embed/t77O9dTbsO8">`

Answers to the Given Questions

(a) Difference Between Padding and Margin in the CSS Box Model

Feature	Padding	Margin
---------	---------	--------

Definition	Space between content and the border of an element	Space between an element and its surrounding elements
Affects Background?	Yes, background color or image extends into padding	No, margin is transparent
Usage	Used to control spacing inside an element	Used to control spacing between elements

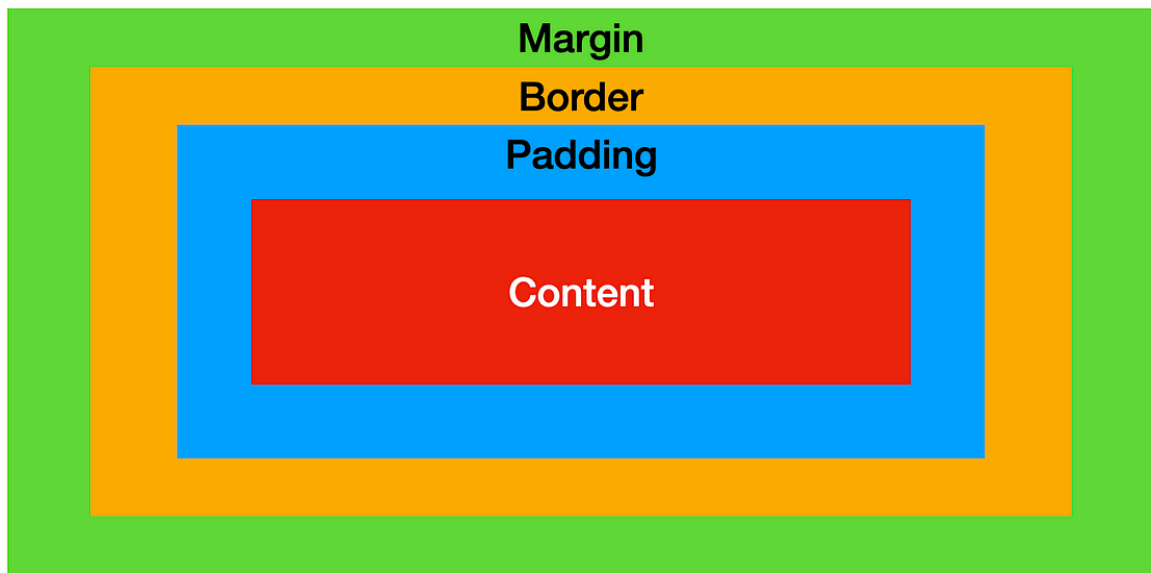
Example:

```

CSS
Copy Edit

.box {
  padding: 10px; /* Space inside the border */
  margin: 20px; /* Space outside the border */
  border: 2px solid black;
}
Explain

```



(b) Differences Between GET and POST Methods in Form Submitting

Feature	GET	POST
Data Visibility	Sent in the URL (visible)	Sent in the request body (hidden)
Security	Less secure	More secure
Data Length Limit?	Limited (depends on URL length)	No size restriction

Use Cases	Retrieving data (e.g., search queries)	Submitting sensitive data (e.g., login, signup)
------------------	--	---

Example:

```
html
<form method="GET" action="search.php">
  <input type="text" name="query">
  <button type="submit">Search</button>
</form>
```

- Here, `search.php?query=value` will appear in the URL.

```
html
<form method="POST" action="login.php">
  <input type="text" name="username">
  <input type="password" name="password">
  <button type="submit">Login</button>
</form>
```

- Here, data is sent securely without appearing in the URL.

(c) What HTML Stands For and Image Tag Based on File Organization

- **HTML:** Stands for **HyperText Markup Language**. It is the standard language for creating web pages.

Image Tag for Different File Locations

1. Both in the Same Folder

html

Copy Edit

```

```

2. `contact.jpg` in a Subfolder Called "images"

html

Copy Edit

```

```

3. `links.html` Inside "otherfiles" Folder, and `contact.jpg` in the Main Directory

html

Copy Edit

```

```

- `../` moves up one directory to access `contact.jpg`.

(d) Difference Between `id` and `class` in HTML

Feature	<code>id</code>	<code>class</code>
---------	-----------------	--------------------

Uniqueness	Must be unique per page	Can be used multiple times
Selector in CSS	<code>#idname</code>	<code>.classname</code>
Use Case	Identifying a single element	Styling multiple elements

Example:

```

html


# PHP database interaction and SQL injection



## Database Interaction with PHP (CRUD Operations & Security)



When working with databases in PHP, we typically perform the following operations:



1. Connecting to the Database
2. Retrieving Data (SELECT)
3. Inserting Data (INSERT)
4. Updating Data (UPDATE)


```

5. Deleting Data (DELETE)

Additionally, we must **protect our application from SQL injection** by using **prepared statements** or **ORMs**.

1. Connecting to a MySQL Database in PHP

PHP provides two main ways to interact with MySQL:

MySQLi (MySQL Improved Extension) - Procedural & Object-Oriented

PDO (PHP Data Objects) - More secure and supports multiple databases

Using MySQLi (Object-Oriented)

```
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "my_database";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
```

Using PDO (Recommended for Security & Flexibility)

```
try {
    $conn = new PDO("mysql:host=localhost;dbname=my_database", "root", "");
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch (PDOException $e) {
    die("Connection failed: " . $e->getMessage());
}
```

2. Retrieving Data (SELECT Query)

We can fetch data in multiple ways:

Using MySQLi (Object-Oriented)

```
$sql = "SELECT id, name, email FROM users";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . " - Name: " . $row["name"] . "<br>";
    }
} else {
    echo "0 results";
}
```

Using PDO (Prepared Statement - More Secure)

```
$stmt = $conn->prepare("SELECT id, name FROM users WHERE id = ?");
$stmt->execute([1]);
$user = $stmt->fetch(PDO::FETCH_ASSOC);
echo "User Name: " . $user['name'];
```

3. Inserting Data (INSERT Query)

Using MySQLi

```
$sql = "INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $conn->error;
}
```

Using PDO (Prepared Statement - Prevents SQL Injection)

```
$stmt = $conn->prepare("INSERT INTO users (name, email) VALUES (?, ?)");
$stmt->execute(["John Doe", "john@example.com"]);
```

```
echo "New record created successfully";
```

4. Updating Data (UPDATE Query)

Using MySQLi

```
$sql = "UPDATE users SET name='Jane Doe' WHERE id=1";
```

```
if ($conn->query($sql) === TRUE) {  
    echo "Record updated successfully";  
} else {  
    echo "Error: " . $conn->error;  
}
```

Using PDO (Prepared Statement)

```
$stmt = $conn->prepare("UPDATE users SET name=? WHERE id=?");  
$stmt->execute(["Jane Doe", 1]);  
echo "Record updated successfully";
```

5. Deleting Data (DELETE Query)

Using MySQLi

```
$sql = "DELETE FROM users WHERE id=1";
```

```
if ($conn->query($sql) === TRUE) {  
    echo "Record deleted successfully";  
} else {  
    echo "Error: " . $conn->error;  
}
```

Using PDO (Prepared Statement)

```
$stmt = $conn->prepare("DELETE FROM users WHERE id=?");  
$stmt->execute([1]);  
echo "Record deleted successfully";
```

6. SQL Injection & Prevention

What is SQL Injection?

SQL Injection is a hacking technique where **malicious SQL code** is inserted into an input field to **manipulate the database**. This can lead to **data leaks, unauthorized access, or data deletion**.

Example of SQL Injection Attack

Imagine a login form where the input is directly inserted into an SQL query:

```
$username = $_GET['username'];  
$password = $_GET['password'];
```

```
$sql = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
```

If an attacker enters:

```
username: ' OR 1=1 --  
password: anything
```

The query becomes:

```
SELECT * FROM users WHERE username = " OR 1=1 --" AND password = 'anything'
```

Since **1=1** is **always true**, the attacker logs in **without knowing the password!**

How to Prevent SQL Injection?

1. Use Prepared Statements (Parameterized Queries)

Using **PDO or MySQLi prepared statements** ensures **user input is treated as data** rather than SQL code.

Secure PDO Example

```
$stmt = $conn->prepare("SELECT * FROM users WHERE username = ? AND password = ?");  
$stmt->execute([$username, $password]);
```

2. Use `mysqli_real_escape_string()`

If you **must** use raw SQL, escape user input:

```
$username = $conn->real_escape_string($_GET['username']);  
$password = $conn->real_escape_string($_GET['password']);  
$sql = "SELECT * FROM users WHERE username='$username' AND password='$password'";
```

3. Limit Database Privileges

- Use a **low-privileged database user**.
- Avoid giving **DROP TABLE** or **DELETE ALL** permissions.

4. Use Web Application Firewalls (WAFs)

- Tools like **ModSecurity** can detect and block SQL injection attacks.

5. Validate & Sanitize User Input

- Allow only **expected characters** (e.g., alphanumeric usernames).
- Use **filtering functions** like `filter_var()` for emails.

```
$email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);
```

7. Best Practices for Secure Database Interaction

Use PDO with Prepared Statements
Use Environment Variables for Credentials
Hash Passwords Using `password_hash()`
Limit Database Privileges
Enable Logging & Monitoring for Suspicious Activity

Conclusion

- **SQL queries** in PHP allow interaction with databases for CRUD operations.
- **SQL Injection** is a serious vulnerability that can expose your database.
- **Prepared statements** are the best way to prevent SQL Injection.
- **Proper validation and escaping** help enhance security.

Would you like a **practical project** demonstrating secure PHP-MySQL interaction?

Bob Modified Code

1. Database.php

```
<?php
$host = "localhost";
$user = "your_username";
$password = "your_password";
$dbname = "your_database";

$conn = mysqli_connect($host, $user, $password, $dbname);

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

2. Login.php


```

<?php
session_start();
include 'database.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $user = mysqli_real_escape_string($conn, $_POST['username']);
    $pass = mysqli_real_escape_string($conn, $_POST['password']);
    $remember = isset($_POST['remember']);

    $sql = "SELECT id FROM userinfo WHERE username = '$user' AND password = '$pass'";
    $result = mysqli_query($conn, $sql);
    $user_data = mysqli_fetch_assoc($result);

    if ($user_data) {
        $_SESSION['user_id'] = $user_data['id']; // Store only user ID in session

        if ($remember) {
            ↓
            setcookie('user_id', $user_data['id'], time() + 86400, "/");
        }
    }
}

```

```

        $_SESSION['user_id'] = $user_data['id']; // Store only user ID in session
    }

    if ($remember) {
        setcookie('user_id', $user_data['id'], time() + 86400, "/");
    }

    header("Location: index.php");
    exit();
} else {
    echo "Invalid username or password!";
}
}
?>

```

```

<form method="post" action="login.php">
    <input type="text" name="username" placeholder="Username" required>
    <input type="password" name="password" placeholder="Password" required>
    <input type="checkbox" name="remember"> Remember Me
    <button type="submit">Log In</button>
</form>

```

3. Index.php

```
1  <?php
2  session_start();
3  include 'database.php';
4
5  if (!isset($_SESSION['user_id']) && !isset($_COOKIE['user_id'])) {
6      header("Location: login.php");
7      exit();
8  }
9
10 $user_id = isset($_SESSION['user_id']) ? $_SESSION['user_id'] : $_COOKIE['user_id'];
11
12 // Fetch username from the database using user ID
13 $sql = "SELECT username FROM userinfo WHERE id = '$user_id'";
14 $result = mysqli_query($conn, $sql);
15 $user_data = mysqli_fetch_assoc($result);
16
17 if (!$user_data) {
18     header("Location: logout.php");
19     exit();
20 }
21 ?>
22
23 <h1>Welcome <?php echo htmlspecialchars($user_data['username']); ?>!</h1>
24 <a href="logout.php">Logout</a>
25
```

4. Logout.php

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

setcookie('user_id', '', time() - 3600, "/");

header("Location: login.php");
exit();
?>
```