Your PHP Guide: Q&A Edition

Unit 1: Introduction to PHP

1. What are the advantages of using PHP for web development?
2. Explain the basic structure of a PHP script.
3. How does PHP handle case sensitivity in variable names and keywords?
4. Differentiate between echo and print in PHP with examples.
5. Write a PHP program to display "Welcome to PHP Programming."

Unit 2: PHP Variables and Data Types

1. What are the different types of data types supported by PHP? Explain each with examples.
2. Explain the difference between global and local variables in PHP.
3. Write a PHP script to calculate the area of a circle using a variable for the radius.

4. How can constants be defined in PHP? Write an example.
5. Explain variable scope in PHP with examples.

Unit 3: Control Structures

1. Write a PHP program to check if a number is even or odd.
2. Explain the if-else and switch statements in PHP with examples.
3. Write a PHP program to display numbers from 1 to 10 using a for loop.
4. How does the while loop differ from the do-while loop in PHP? Explain with examples.
5. Write a program to find the largest number among three user inputs using PHP.

Unit 4: Functions in PHP

1. What is the difference between user-defined and built-in functions in PHP?
2. Write a user-defined function to calculate the factorial of a number in PHP.

3. Explain the difference between passing arguments by value and by reference with examples.
4. List any five string functions in PHP and explain their usage with examples.
5. Write a PHP script using the date() function to display the current date and time.

Unit 5: Arrays in PHP

1. Differentiate between indexed, associative, and multidimensional arrays in PHP.
2. Write a PHP script to merge two arrays using array functions.
3. How can you iterate through an associative array in PHP? Write a program to demonstrate this.
4. Explain the array_push() and array_pop() functions in PHP with examples.
5. Write a PHP program to sort an array in ascending order.

Unit 6: File Handling

1. How can you open, read, write, and close files in PHP? Explain with examples.
2. Write a PHP script to read the contents of a text file and display it on a webpage.
3. Explain the difference between the fopen() and file_get_contents() functions in PHP.
4. Write a PHP program to append data to an existing file.
5. How can you handle errors while working with files in PHP?

Unit 7: PHP and MySQL

1. Write a PHP script to connect to a MySQL database.
2. How can you fetch data from a MySQL table using PHP? Explain with an example.
3. Write a PHP program to insert user input into a MySQL database.
4. Explain the concept of prepared statements in PHP with examples.
5. What is the difference between mysqli and PDO in PHP?

Unit 8: Advanced Topics

1. What is a session in PHP? How is it different from a cookie?
2. Write a PHP program to create and destroy a session.
3. Explain the concept of middleware in PHP frameworks.
4. How can you implement form validation in PHP? Write a program to validate a login form.
5. What is the importance of PHP security measures? Explain SQL injection and how to prevent it.

ANSWERAS WITH EXPLANATION

Unit 1: Introduction to PHP

1. What are the advantages of using PHP for web development?

PHP is widely used for server-side web development due to its flexibility and power. Key advantages include:

- **Open Source**: PHP is free to use and has a large community.

- **Cross-Platform**: It runs on all major operating systems, including Windows, Linux, and macOS.
- **Ease of Learning**: Its syntax is simple and similar to C and Java, making it easy for beginners.
- **Compatibility**: PHP integrates seamlessly with popular databases like MySQL, PostgreSQL, and MongoDB.
- **Wide Framework Support**: Frameworks like Laravel, CodeIgniter, and Symfony make development faster and more efficient.
- **Scalability**: It supports small websites to large-scale enterprise applications.

---

2. Explain the basic structure of a PHP script.
A PHP script is typically embedded within HTML and starts with <?php and ends with ?>. Example:

```
<!DOCTYPE html>

<html>

<head>

    <title>PHP Example</title>

</head>

<body>
```

```
  <?php
    echo "Hello, World!";
  ?>
</body>
</html>
```

- <?php opens the PHP code block.
- echo outputs text or HTML.
- ?> closes the PHP code block.

---

3. How does PHP handle case sensitivity in variable names and keywords?

- **Variables**: Case-sensitive. $Name and $name are different.
- **Keywords and Functions**: Not case-sensitive. echo, ECHO, and Echo work the same.
  Example:

```
<?php
  $name = "John";
  echo $name; // Valid
  echo $Name; // Undefined variable error
```

```
?>
```

---

4. Differentiate between echo and print in PHP with examples.

- **echo**: Faster, can output multiple strings separated by commas, doesn’t return a value.
- **print**: Slightly slower, can output only one string, returns 1 on success.
  Example:

```
<?php
    echo "Hello", " World!"; // Outputs: Hello World!
    print "Hello World!";    // Outputs: Hello World!
?>
```

---

5. Write a PHP program to display "Welcome to PHP Programming."

```
<?php
    echo "Welcome to PHP Programming.";
?>
```

Output: Welcome to PHP Programming.

Unit 2: PHP Variables and Data Types

1. What are the different types of data types supported by PHP? Explain each with examples.
PHP supports several data types:

1. **String**: Sequence of characters.

   ```
   $string = "Hello, PHP!";
   ```

   echo $string; // Output: Hello, PHP!

2. **Integer**: Non-decimal numbers.

   ```
   $integer = 25;
   ```

   echo $integer; // Output: 25

3. **Float/Double**: Decimal numbers.

   ```
   $float = 3.14;
   ```

   echo $float; // Output: 3.14

4. **Boolean**: True or false.

   ```
   $is_php_easy = true;
   ```

   echo $is_php_easy; // Output: 1 (true)

5. **Array**: Collection of values.

   ```
   $array = array("HTML", "CSS", "PHP");
   ```

```
echo $array[2]; // Output: PHP
```

6. **Object**: Instances of a class.

```
class Car {
    public $model = "Tesla";
}
$car = new Car();
echo $car->model; // Output: Tesla
```

7. **Null**: A variable with no value.

```
$var = null;
echo $var; // Output: nothing
```

8. **Resource**: Special handlers for files or database connections.

---

2. Explain the difference between global and local variables in PHP.

- **Global Variables**: Declared outside a function and accessed using the global keyword or $GLOBALS.
- **Local Variables**: Declared inside a function and accessible only within that function.
  Example:

```
<?php
    $globalVar = "I am global";
    function test() {
        global $globalVar;
        $localVar = "I am local";
        echo $globalVar; // Access global variable
        echo $localVar;  // Access local variable
    }
    test();
    echo $localVar; // Error: Undefined variable
?>
```

---

3. Write a PHP script to calculate the area of a circle using a variable for the radius.

```
<?php
    $radius = 5;
    $area = pi() * $radius * $radius;
    echo "The area of the circle with radius $radius is
$area.";
```

```
?>
```

Output: The area of the circle with radius 5 is 78.539816339744.

---

4. How can constants be defined in PHP? Write an example.
Constants are defined using the define() function or the const keyword.
Example using define():

```
<?php
   define("PI", 3.14);
   echo PI; // Output: 3.14
?>
```

Example using const:

```
<?php
   const GREETING = "Welcome!";
   echo GREETING; // Output: Welcome!
?>
```

- Constants are global and cannot be changed once set.

5. Explain variable scope in PHP with examples.
Variable scope determines where a variable can be accessed.

1. **Local Scope**: Inside a function.
2. **Global Scope**: Outside all functions.
3. **Static Scope**: Retains value between function calls.
4. **Super Global Scope**: Predefined variables like $_POST, $_GET.
   Example:

```
<?php

function countCalls() {

    static $count = 0;

    $count++;

    echo $count;

}

countCalls(); // Output: 1

countCalls(); // Output: 2

?>
```

Unit 3: Control Structures

1. Write a PHP program to check if a number is even or odd.

```
<?php
    $number = 7;
    if ($number % 2 == 0) {
        echo "$number is even.";
    } else {
        echo "$number is odd.";
    }
?>
```

Output: 7 is odd.

---

2. Explain the if-else and switch statements in PHP with examples.

- **if-else**: Executes a block of code based on a condition.
  Example:

```
<?php
    $age = 20;
    if ($age >= 18) {
        echo "You are eligible to vote.";
    } else {
        echo "You are not eligible to vote.";
    }
?>
```

- **switch**: Executes a block of code based on multiple conditions.
  Example:

```
<?php
    $day = "Monday";
    switch ($day) {
        case "Monday":
            echo "Start of the workweek.";
            break;
        case "Friday":
            echo "Weekend is near.";
            break;
        default:
            echo "Regular day.";
    }
?>
```

Output: Start of the workweek.

---

3. Write a PHP program to display numbers from 1 to 10 using a for loop.

```
<?php
    for ($i = 1; $i <= 10; $i++) {
        echo $i . " ";
    }
?>
```

Output: 1 2 3 4 5 6 7 8 9 10

---

4. How does the while loop differ from the do-while loop in PHP? Explain with examples.

- **while loop**: Executes the block as long as the condition is true.
- **do-while loop**: Executes the block at least once before checking the condition.

Examples:

```
// while loop
<?php
    $i = 1;
    while ($i <= 5) {
        echo $i . " ";
        $i++;
    }
?>
```

Output: 1 2 3 4 5

```
// do-while loop
<?php
    $i = 6;
    do {
        echo $i . " ";
        $i++;
    } while ($i <= 5);
?>
```

Output: 6 (runs once even though the condition is false).

5. Write a program to find the largest number among three user inputs using PHP.

```
<?php
    $a = 12;
    $b = 25;
    $c = 9;
    if ($a > $b && $a > $c) {
        echo "$a is the largest.";
    } elseif ($b > $a && $b > $c) {
        echo "$b is the largest.";
    } else {
        echo "$c is the largest.";
    }
?>
```

Output: 25 is the largest.

Unit 4: Functions in PHP

1. What are functions in PHP? Explain the difference between built-in and user-defined functions.

- A **function** is a reusable block of code that performs a specific task.
- **Built-in Functions**: Predefined in PHP, such as strlen(), count().
  Example:

  ```
  <?php
      echo strlen("Hello PHP!"); // Output: 9
  ?>
  ```

- **User-Defined Functions**: Created by users to perform custom tasks.
  Example:

  ```
  <?php
      function greet($name) {
          return "Hello, $name!";
      }
      echo greet("John"); // Output: Hello, John!
  ?>
  ```

---

2. Write a program to create and call a user-defined function to calculate the factorial of a number.

```
<?php
```

```
    function factorial($num) {
        $factorial = 1;
        for ($i = 1; $i <= $num; $i++) {
            $factorial *= $i;
        }
        return $factorial;
    }
    echo "Factorial of 5 is: " . factorial(5);
?>
```

Output: Factorial of 5 is: 120

---

3. What is the difference between passing arguments by value and by reference in PHP?

- **By Value**: A copy of the value is passed. Changes in the function do not affect the original variable. Example:

  ```
  <?php
      function increment($x) {
          $x++;
      }
  ```

```
    $num = 5;

    increment($num);

    echo $num; // Output: 5

?>
```

- **By Reference**: A reference to the variable is passed. Changes in the function affect the original variable.
  Example:

```
<?php

    function increment(&$x) {

        $x++;

    }

    $num = 5;

    increment($num);

    echo $num; // Output: 6

?>
```

---

4. Write a PHP program to demonstrate the use of default arguments in functions.

```
<?php
```

```
    function greet($name = "Guest") {
        return "Hello, $name!";
    }
    echo greet(); // Output: Hello, Guest!
    echo greet("Alice"); // Output: Hello, Alice!
?>
```

---

5. Write a PHP program to calculate the sum of an array using a function.

```
<?php
    function calculateSum($array) {
        $sum = 0;
        foreach ($array as $value) {
            $sum += $value;
        }
        return $sum;
    }
    $numbers = [1, 2, 3, 4, 5];
    echo "The sum of the array is: " .
calculateSum($numbers);

?>
```

Output: The sum of the array is: 15

---

6. Explain recursive functions in PHP with an example.

- A recursive function is a function that calls itself to solve smaller instances of the same problem. Example:

```
<?php

    function factorial($num) {

        if ($num <= 1) {

            return 1;

        }

        return $num * factorial($num - 1);

    }

   echo "Factorial of 5 is: " . factorial(5);

?>
```

Output: Factorial of 5 is: 120

Unit 5: Arrays in PHP

1. What are arrays in PHP? Explain the different types of arrays with examples.

An **array** in PHP is a data structure that stores multiple values in a single variable. PHP supports three types of arrays:

1. **Indexed Array**: Uses numeric keys.

   ```
   <?php
     $fruits = array("Apple", "Banana", "Orange");
     echo $fruits[0]; // Output: Apple
   ?>
   ```

2. **Associative Array**: Uses named keys.

   ```
   <?php
     $ages = array("John" => 25, "Alice" => 30);
     echo $ages["Alice"]; // Output: 30
   ?>
   ```

3. **Multidimensional Array**: Contains other arrays as elements.

   ```
   <?php
   ```

```
        $students = array(
            array("John", 85, "A"),
            array("Alice", 90, "A+")
        );
        echo $students[1][0]; // Output: Alice
    ?>
```

---

2. Write a program to demonstrate traversing an indexed array using a foreach loop.

```
<?php
    $colors = array("Red", "Green", "Blue");
    foreach ($colors as $color) {
        echo $color . " ";
    }
?>
```

Output: Red Green Blue

---

3. Write a program to add key-value pairs to an associative array and display its elements.

```
<?php
    $person = array();
    $person["Name"] = "John";
    $person["Age"] = 25;
    $person["City"] = "New York";
    foreach ($person as $key => $value) {
        echo "$key: $value<br>";
    }
?>
```

Output:

```
Name: John
Age: 25
City: New York
```

---

4. How can you sort an array in PHP? Explain with examples.

PHP provides several functions for sorting arrays:

1. **sort()**: Sorts an indexed array in ascending order.

```
<?php
```

```
    $numbers = array(5, 3, 8, 1);
    sort($numbers);
    print_r($numbers); // Output: Array ( [0] => 1 [1] => 3 [2] => 5 [3] => 8 )
?>
```

2. **rsort()**: Sorts an indexed array in descending order.

```
<?php
    rsort($numbers);
    print_r($numbers); // Output: Array ( [0] => 8 [1] => 5 [2] => 3 [3] => 1 )
?>
```

3. **asort()**: Sorts an associative array by values in ascending order.

```
<?php
    $ages = array("John" => 25, "Alice" => 30);
    asort($ages);
    print_r($ages); // Output: Array ( [John] => 25 [Alice] => 30 )
?>
```

4. **ksort()**: Sorts an associative array by keys in ascending order.

```
<?php
    ksort($ages);
    print_r($ages); // Output: Array ( [Alice] => 30
[John] => 25 )
?>
```

---

5. Write a PHP program to merge two arrays.

```
<?php
    $array1 = array("Red", "Green");
    $array2 = array("Blue", "Yellow");
   $mergedArray = array_merge($array1, $array2);
   print_r($mergedArray);
?>
```

Output:

```
Array ( [0] => Red [1] => Green [2] => Blue [3] => Yellow
)
```

---

6. Write a program to find the maximum and minimum values in an array.

```
<?php
    $numbers = array(10, 5, 8, 20);
    echo "Maximum: " . max($numbers) . "<br>";
    echo "Minimum: " . min($numbers);
?>
```

Output:

```
Maximum: 20
Minimum: 5
```

---

7. Explain how to handle multidimensional arrays in PHP with an example.
Multidimensional arrays are arrays containing other arrays.
Example:

```
<?php
    $matrix = array(
        array(1, 2, 3),
        array(4, 5, 6),
        array(7, 8, 9)
    );
    foreach ($matrix as $row) {
        foreach ($row as $element) {
            echo $element . " ";
        }
        echo "<br>";
    }
?>
```

Output:

```
1 2 3
4 5 6
7 8 9
```

Unit 6: File Handling in PHP

1. What is file handling in PHP? Why is it important?

File handling in PHP involves performing operations like creating, reading, writing, and deleting files on the server.

Importance:

- To store and retrieve data.

- To manage logs or store user inputs.
- To create dynamic websites (e.g., configuration files or content updates).

---

2. Write a PHP program to create a new file and write some text into it.

```
<?php
    $file = fopen("example.txt", "w"); // Open file for writing
    $content = "This is an example of file handling in PHP.\n";
    fwrite($file, $content); // Write to file
    fclose($file); // Close the file
    echo "File created and text written successfully.";
?>
```

Output: File created and text written successfully.

3. Write a PHP program to read the contents of a file.

```
<?php
    $file = fopen("example.txt", "r"); // Open file for reading
    while (!feof($file)) { // Loop until the end of the file
        echo fgets($file) . "<br>"; // Read line by line
    }
    fclose($file); // Close the file
?>
```

Output:

4. Explain the difference between fopen(), fread(), fwrite(), and fclose() in PHP.

1. **fopen()**: Opens a file in a specific mode (e.g., read, write, append).
   Syntax: fopen(filename, mode)
   Example: fopen("example.txt", "r");
2. **fread()**: Reads data from a file.
   Syntax: fread(file, length)
   Example:

   ```
   $content = fread($file, filesize("example.txt"));
   ```
3. **fwrite()**: Writes data to a file.
   Syntax: fwrite(file, string)
   Example: fwrite($file, "Hello, World!");
4. **fclose()**: Closes an open file.
   Syntax: fclose(file)

5. Write a PHP program to append data to an existing file.

```
<?php
    $file = fopen("example.txt", "a"); // Open file in
append mode
    $newContent = "Adding a new line to the file.\n";
    fwrite($file, $newContent); // Append data
    fclose($file); // Close the file
    echo "Data appended successfully.";
?>
```

6. Write a PHP program to delete a file.

```
<?php
    $filename = "example.txt";
    if (file_exists($filename)) {
        unlink($filename); // Delete the file
        echo "File '$filename' deleted successfully.";
    } else {
        echo "File does not exist.";
    }
?>
```

```
Output: File 'example.txt' deleted successfully.
```

---

7. How can you check if a file exists in PHP? Write an example.

You can use the **file_exists()** function to check if a file exists.

Example:

```php
<?php
    $filename = "example.txt";
    if (file_exists($filename)) {
        echo "The file '$filename' exists.";
    } else {
        echo "The file '$filename' does not exist.";
    }
?>
```

```
Output: The file 'example.txt' does not exist.
```

---

8. Write a PHP program to read a file using file_get_contents() and display its contents.

```php
<?php
```

```
    $filename = "example.txt";
    if (file_exists($filename)) {
        $content = file_get_contents($filename);
        echo $content;
    } else {
        echo "File not found.";
    }
?>
```

Output (if file exists):

Unit 7: Cookies and Sessions in PHP

1. What are cookies in PHP? Why are they used?

Cookies are small pieces of data stored on the client's browser by the server. They are used to store user preferences, session details, and other data that persists across multiple page requests.

Example Use Cases:

- Remembering login details.
- Storing user preferences like language or theme.
- Tracking user activity for analytics.

2. Write a PHP program to set and retrieve a cookie.

Setting a Cookie:

```
<?php
    setcookie("username", "LOPA", time() + (86400 *
7), "/"); // Valid for 7 days
    echo "Cookie 'username' has been set.";
?>
```

Retrieving a Cookie:

```
<?php
    if (isset($_COOKIE["username"])) {
        echo "Username: " . $_COOKIE["username"];
    } else {
        echo "Cookie 'username' is not set.";
    }
?>
```

Output (if cookie exists): Username: LOPA

3. How can you delete a cookie in PHP? Write an example.
To delete a cookie, set its expiration time to a past time.

```
<?php
    setcookie("username", "", time() - 3600, "/"); //
Expire the cookie
    echo "Cookie 'username' has been deleted.";
?>
```

---

4. What are sessions in PHP? How are they different from cookies?
Sessions store user data on the server instead of the client’s browser.

Differences:

Aspect	Cookies	Sessions
Storage	Client-side (browser).	Server-side.
Security	Less secure (data visible to users).	More secure (data stored on server).

Aspect	Cookies	Sessions
Capacity	Limited storage size.	No size limit.
Lifetime	Persist across browser sessions (if set).	Destroyed when the browser closes (by default).

---

5. Write a PHP program to start a session and store session variables.

```
<?php
    session_start(); // Start the session
    $_SESSION["username"] = "LOPA";
    $_SESSION["role"] = "Admin";
     echo "Session variables are set.";
?>
```

---

6. Write a PHP program to retrieve session variables.

```
<?php
```

```
    session_start(); // Resume the session


    if (isset($_SESSION["username"])) {
        echo "Username: " . $_SESSION["username"] .
"<br>";
        echo "Role: " . $_SESSION["role"];
    } else {
        echo "Session variables are not set.";
    }
?>
```

Output:

makefile

Copy code

```
Username: LOPA
Role: Admin
```

---

7. Write a PHP program to destroy a session.

```
<?php
    session_start(); // Resume the session
    session_unset(); // Remove all session variables

    session_destroy(); // Destroy the session

    echo "Session has been destroyed.";

?>
```

Output: Session has been destroyed.

---

8. What is the role of session_start() in PHP?
The **session_start()** function initializes a session or resumes the current one. It is required to access or modify session variables.

---

9. Write a PHP program to demonstrate the difference between cookies and sessions.

```
<?php

    // Cookie Example

    setcookie("user", "CookieUser", time() + 3600, "/");
// 1-hour cookie

    // Session Example

    session_start();

    $_SESSION["user"] = "SessionUser";
    // Display values
    echo "Cookie Value: " . $_COOKIE["user"] . "<br>";
    echo "Session Value: " . $_SESSION["user"];
?>
```

Output:

```
Cookie Value: CookieUser
Session Value: SessionUser
```

---

10. What are some common use cases of cookies and sessions?

Cookies	**Sessions**
Storing user preferences like theme or language.	Managing logged-in user details.
Tracking user activity for analytics.	Shopping cart functionality in e-commerce apps.
Storing non-sensitive data for a longer duration.	Temporary storage of sensitive data.

Unit 8: Database Connectivity in PHP

1. What is database connectivity in PHP? Why is it important?

Database connectivity in PHP refers to the process of connecting a PHP application to a database (such as MySQL, PostgreSQL, etc.) to interact with and manipulate data.

Importance:

- Allows dynamic interaction with data (CRUD operations).
- Essential for creating data-driven applications like e-commerce sites, blogs, and user management systems.
- Ensures persistence of data beyond page reloads.

---

2. Explain the MySQLi and PDO methods for database connectivity in PHP.

PHP offers two primary methods for interacting with databases: **MySQLi** and **PDO**.

MySQLi (MySQL Improved)

- Can be used with MySQL databases only.
- Provides both procedural and object-oriented interfaces.

- Supports prepared statements, transactions, and more.

PDO (PHP Data Objects)

- Supports multiple databases (MySQL, PostgreSQL, SQLite, etc.).
- Provides only an object-oriented interface.
- Supports prepared statements and transactions.

---

3. Write a PHP program to connect to a MySQL database using MySQLi (Procedural).

```
<?php
    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "testdb";
    // Create connection
    $conn = mysqli_connect($servername, $username,
$password, $dbname);
    // Check connection
    if (!$conn) {
        die("Connection failed: " .
mysqli_connect_error());

    }

   echo "Connected successfully";

?>
```

Output: Connected successfully

---

4. Write a PHP program to connect to a MySQL database using PDO.

```
<?php

    $servername = "localhost";

    $username = "root";

    $password = "";

    $dbname = "testdb";

 try {

        $conn = new
PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);

        $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
```

```
        echo "Connected successfully";
    } catch (PDOException $e) {
        echo "Connection failed: " . $e->getMessage();
    }
?>
```

Output: Connected successfully

---

5. How do you perform a SELECT query in PHP using MySQLi?

```
<?php
    $conn = mysqli_connect("localhost", "root", "",
"testdb");
     $sql = "SELECT id, name, email FROM users";
    $result = mysqli_query($conn, $sql);
     if (mysqli_num_rows($result) > 0) {
        // Output data of each row
        while($row = mysqli_fetch_assoc($result)) {
            echo "id: " . $row["id"]. " - Name: " .
$row["name"]. " - Email: " . $row["email"]. "<br>";
        }
    } else {
        echo "0 results";
    }
    mysqli_close($conn);
?>
```

Output (if records exist):

```
id: 1 - Name: John - Email: john@example.com
id: 2 - Name: Jane - Email: jane@example.com
```

---

6. How do you perform a SELECT query in PHP using PDO?

```
<?php
    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "testdb";

    try {
        $conn = new
PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);

        $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);


        $stmt = $conn->prepare("SELECT id, name, email
FROM users");

        $stmt->execute();

        // Set the resulting array to associative

        $result = $stmt-
>fetchAll(PDO::FETCH_ASSOC);

       foreach ($result as $row) {

           echo "id: " . $row["id"]. " - Name: " .
$row["name"]. " - Email: " . $row["email"]. "<br>";

        }

    } catch (PDOException $e) {

        echo "Error: " . $e->getMessage();

    }

      $conn = null;

?>
```

Output (if records exist):

```
id: 1 - Name: John - Email: john@example.com

id: 2 - Name: Jane - Email: jane@example.com
```

---

7. How do you perform an INSERT query in PHP using MySQLi?

```
<?php

   $conn = mysqli_connect("localhost", "root", "",
"testdb");

   $name = "John";

   $email = "john@example.com";

 $sql = "INSERT INTO users (name, email) VALUES
('$name', '$email')";

 if (mysqli_query($conn, $sql)) {

       echo "New record created successfully";

   } else {

       echo "Error: " . $sql . "<br>" .
mysqli_error($conn);

   }

   mysqli_close($conn);
```

```
?>
```

Output: New record created successfully

---

8. How do you perform an INSERT query in PHP using PDO?

```
<?php
    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "testdb";
     try {
        $conn = new
PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);
        $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
 $name = "John";
        $email = "john@example.com";
        $stmt = $conn->prepare("INSERT INTO users
(name, email) VALUES (:name, :email)");
```

```
        $stmt->bindParam(':name', $name);
        $stmt->bindParam(':email', $email);
        $stmt->execute();
        echo "New record created successfully";
     } catch (PDOException $e) {
        echo "Error: " . $e->getMessage();
    }
 $conn = null;
?>
```

Output: New record created successfully

---

9. How do you perform an UPDATE query in PHP using MySQLi?

```
<?php
    $conn = mysqli_connect("localhost", "root", "",
"testdb");
    $id = 1;
    $name = "John Doe";
    $email = "johndoe@example.com";
    $sql = "UPDATE users SET name='$name',
email='$email' WHERE id=$id";

    if (mysqli_query($conn, $sql)) {

        echo "Record updated successfully";

    } else {

        echo "Error: " . $sql . "<br>" .
mysqli_error($conn);

    }

    mysqli_close($conn);

?>
```

Output: Record updated successfully

---

10. How do you perform an UPDATE query in PHP using PDO?

```
<?php

    $servername = "localhost";

    $username = "root";

    $password = "";

    $dbname = "testdb";
    try {
        $conn = new
PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);
        $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
        $id = 1;
        $name = "John Doe";
        $email = "johndoe@example.com";
        $stmt = $conn->prepare("UPDATE users SET
name=:name, email=:email WHERE id=:id");
        $stmt->bindParam(':id', $id);
        $stmt->bindParam(':name', $name);
        $stmt->bindParam(':email', $email);
        $stmt->execute();
        echo "Record updated successfully";
 } catch (PDOException $e) {
        echo "Error: " . $e->getMessage();
    }
 $conn = null;
```

```
?>
```

Output: Record updated successfully

Unit 9: Object-Oriented Programming (OOP) in PHP

1. What is Object-Oriented Programming (OOP)? Explain its basic concepts.
Object-Oriented Programming (OOP) is a programming paradigm based on the concept of **objects**, which can contain data (in the form of fields, often known as attributes or properties) and code (in the form of procedures, often known as methods).

Basic Concepts of OOP:

- **Classes**: A blueprint for creating objects, providing initial values for state (variables) and implementations of behavior (functions/methods).
- **Objects**: Instances of classes. They represent entities in a program.
- **Encapsulation**: The bundling of data with the methods that operate on that data. It restricts direct access to some of an object's components, which can prevent unintended interference and misuse of the data.

- **Inheritance**: A mechanism where a new class (child class) inherits the properties and methods of an existing class (parent class).
- **Polymorphism**: The ability of different objects to respond to the same method in different ways.
- **Abstraction**: Hiding the complexity of a system and exposing only the essential parts to the user.

---

2. What is the difference between a class and an object in PHP?

- **Class**: A class is a blueprint for creating objects. It defines properties (variables) and methods (functions) that the created objects will have.

  Example:

```
class Car {
    public $color;
    public $model;
    public function displayInfo() {
        echo "Color: " . $this->color . " Model: " .
$this->model;
    }
```

}

- **Object**: An object is an instance of a class. It is created based on the structure defined by the class.

  Example:

  ```
  $myCar = new Car();  // Creating an object

  $myCar->color = "Red";  // Setting the property

  $myCar->model = "Toyota"; // Setting the property

  $myCar->displayInfo();  // Calling the method
  ```

---

3. What is encapsulation in PHP?
Encapsulation refers to the bundling of data (properties) and methods (functions) that operate on the data into a single unit called a **class**. It also restricts access to some of the object's components, which can prevent unintended interference and misuse.

Access Modifiers:

- **public**: The property or method can be accessed from anywhere.
- **private**: The property or method can only be accessed within the class.
- **protected**: The property or method can be accessed within the class and by subclasses.

Example:

```
class Car {
    private $model;
    private $color;
    public function setModel($model) {
        $this->model = $model;
    }
    public function getModel() {
        return $this->model;
    }
    public function setColor($color) {
        $this->color = $color;
    }
    public function getColor() {
        return $this->color;
    }
}
```

4. What is inheritance in PHP?
Inheritance allows a class to acquire the properties and methods of another class. It helps in reusing the code, creating a hierarchy of classes, and extending functionality without rewriting the code.

Example:

```
class Vehicle {

    public $color;

   public function setColor($color) {

       $this->color = $color;

    }

   public function getColor() {

       return $this->color;

    }

}

class Car extends Vehicle {  // Car class inherits from
Vehicle

    public $model;

   public function setModel($model) {

       $this->model = $model;

    }
    public function getModel() {
        return $this->model;
    }
}
$myCar = new Car();
$myCar->setColor("Red");
$myCar->setModel("Toyota");
echo $myCar->getColor();  // Red
echo $myCar->getModel();  // Toyota
```

---

5. What is polymorphism in PHP?
Polymorphism allows objects of different classes to be treated as objects of a common parent class. It also allows methods in different classes to have the same name but behave differently.

Method Overriding: A subclass can override a method defined in a parent class.

Example:

```
class Animal {
    public function makeSound() {
        echo "Animal makes a sound";
    }
}
 class Dog extends Animal {
    public function makeSound() {
        echo "Dog barks";
    }
}
 class Cat extends Animal {
    public function makeSound() {
        echo "Cat meows";
    }
}
$dog = new Dog();
$dog->makeSound();  // Dog barks
 $cat = new Cat();
$cat->makeSound();  // Cat meows
```

6. What is abstraction in PHP?
Abstraction is the concept of hiding the implementation details of a system and exposing only the essential features to the user. It helps in reducing complexity and allows the programmer to focus on interactions rather than implementation.

In PHP, abstraction can be achieved using **abstract classes** and **abstract methods**.

Example:

```
abstract class Animal {
    abstract public function sound();
}
class Dog extends Animal {
    public function sound() {
        echo "Bark";
    }
}
$dog = new Dog();
$dog->sound();  // Bark
```

7. Write a PHP program to create a simple class and object.

```
<?php
class Car {
    public $make;
    public $model;
    public function displayInfo() {
        echo "Make: " . $this->make . " Model: " . $this->model;
    }
}
 $myCar = new Car();
$myCar->make = "Toyota";
$myCar->model = "Corolla";
$myCar->displayInfo();
?>
```

Output: Make: Toyota Model: Corolla

---

8. Write a PHP program to demonstrate inheritance.

```php
<?php
class Vehicle {
    public $color;
 public function setColor($color) {
        $this->color = $color;
    }
 public function getColor() {
        return $this->color;
    }
}
 class Car extends Vehicle {
    public $model;
 public function setModel($model) {
        $this->model = $model;
    }
     public function getModel() {
        return $this->model;
    }
}
```

```
 $myCar = new Car();
$myCar->setColor("Blue");
$myCar->setModel("Honda Civic");
 echo "Color: " . $myCar->getColor();  // Blue
echo "Model: " . $myCar->getModel();  // Honda Civic
?>
```

---

9. What are interfaces in PHP? How do they differ from abstract classes?

Interfaces define a contract that any class implementing the interface must adhere to. An interface can only declare methods but cannot provide their implementation. In contrast, an **abstract class** can declare methods and provide implementation for some of them.

Example of Interface:

```
interface Vehicle {
    public function start();
    public function stop();
}
 class Car implements Vehicle {
    public function start() {
        echo "Car is starting";
    }
 public function stop() {
        echo "Car is stopping";
    }
}
```

---

10. Write a PHP program to demonstrate the use of polymorphism.

```
<?php
class Shape {
    public function draw() {
        echo "Drawing a shape";
    }
}
 class Circle extends Shape {
    public function draw() {
        echo "Drawing a circle";
    }
}
 class Square extends Shape {
    public function draw() {
        echo "Drawing a square";
    }
}
$circle = new Circle();
$circle->draw();  // Drawing a circle
 $square = new Square();
$square->draw();  // Drawing a square
?>
```

Unit 10: Error Handling in PHP

1. What is error handling in PHP? Why is it important?

Error handling in PHP refers to the process of responding to and managing runtime errors that may occur during the execution of a script. Proper error handling is crucial as it helps to:

- Detect and manage errors efficiently, ensuring that the program doesn't crash.
- Provide meaningful error messages to users or developers to help diagnose issues.
- Maintain the stability and reliability of applications by preventing the leakage of sensitive information.

In PHP, error handling is done using error handling functions, try-catch blocks, and custom error handling functions.

---

2. What are the different types of errors in PHP?
PHP defines several types of errors:

- **Parse Errors**: Occur when there are syntax issues in the code (e.g., missing semicolons or parentheses). Example:

  ```
  // Missing semicolon causes a parse error
  echo "Hello, World"
  ```

- **Fatal Errors**: Occur when PHP cannot execute the code due to a serious issue, such as calling an undefined function or class. Example:

  ```
  // Fatal error: Call to undefined function
  nonExistingFunction()
  ```

```
nonExistingFunction();
```

- **Warning Errors**: Non-fatal errors that are logged but don't stop the script from executing. Often caused by issues like file not found. Example:

  ```
  // Warning: include(): Failed opening required 'file.php'

  include('file.php');
  ```

- **Notice Errors**: The least severe errors, often related to undefined variables or array indices that are not initialized. Example:

  ```
  // Notice: Undefined variable: undefinedVar

  echo $undefinedVar;
  ```

- **Deprecated Errors**: Occur when using features or functions that are no longer supported in the current PHP version. Example:

  ```
  // Deprecated: Function create_function() is deprecated

  $func = create_function('$a', 'return $a+1;');
  ```

---

3. How can you handle errors in PHP using try-catch?

The try-catch block is used to catch exceptions thrown by the code and handle them gracefully.

Syntax:

```
try {
    // Code that may throw an exception
    $result = divide(10, 0);  // This will throw an exception
} catch (Exception $e) {
    // Code to handle the exception
    echo "Error: " . $e->getMessage();  // Print exception message
}
```

Example:

```
<?php
function divide($a, $b) {
    if ($b == 0) {
        throw new Exception("Division by zero");
    }
    return $a / $b;
}
```

```
try {
    echo divide(10, 0);  // Throws an exception
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();  // Catches and handles the error
}
?>
```

Output: Error: Division by zero

---

4. What is the purpose of set_error_handler() function in PHP?

The set_error_handler() function in PHP allows you to define a custom function to handle errors. It replaces the default PHP error handler with the custom one you provide.

Syntax:

```
set_error_handler("errorHandlerFunction");
```

Example:

```
<?php
// Custom error handler function
function customError($errno, $errstr) {
    echo "Error [$errno]: $errstr<br>";

}

// Set the custom error handler

set_error_handler("customError");

// Trigger an error

echo 10 / 0;  // Division by zero error

?>

Output: Error [2]: Division by zero
```

---

5. What are the different error levels in PHP?
PHP defines several error reporting levels that determine which types of errors should be reported.

- **E_ERROR**: Fatal run-time errors that stop the script.
- **E_WARNING**: Run-time warnings that do not stop script execution.
- **E_NOTICE**: Run-time notices, which are less severe.
- **E_PARSE**: Compile-time parse errors.
- **E_DEPRECATED**: Warnings about the usage of deprecated features.

- **E_ALL**: Reports all types of errors, including notices, warnings, and other errors.

Example of enabling error reporting:

```
// Report all errors

error_reporting(E_ALL);

// Display errors on the screen

ini_set("display_errors", 1);
```

---

6. How can you log errors in PHP?
You can log errors in PHP by enabling error logging in your PHP configuration and using the error_log() function to log specific errors.

Enable Error Logging: In the php.ini file:

```
log_errors = On

error_log = /path/to/error_log
```

Using error_log() function:

```
<?php

// Custom error logging

error_log("This is a custom error message.", 3,
"/path/to/error_log");
```

```
?>
```

This will log the error message to the specified file.

---

7. How do you handle fatal errors in PHP?
Fatal errors cannot be caught using try-catch. However, you can handle fatal errors using a custom shutdown function and register_shutdown_function().

Example:

```
<?php

// Custom shutdown function to handle fatal errors

function shutdownFunction() {

    $error = error_get_last();

    if ($error !== NULL) {

        echo "Fatal error: " . $error['message'];

    }

}

// Register the shutdown function

register_shutdown_function("shutdownFunction");

// Trigger a fatal error

echo $undefinedVar;
```

```
?>
```

This will catch fatal errors and print a custom message.

---

8. What is the difference between throw and trigger_error() in PHP?

- **throw**: Used to throw exceptions, which can be caught by try-catch blocks.
- **trigger_error()**: Used to trigger a user-level error or warning.

Example of throw:

```
<?php

throw new Exception("An error occurred.");

?>
```

Example of trigger_error():

```
<?php

trigger_error("This is a custom warning.",
E_USER_WARNING);

?>
```

---

9. Write a PHP program to handle division by zero using try-catch.

```
<?php
function divide($a, $b) {
    if ($b == 0) {
        throw new Exception("Cannot divide by zero");
    }
    return $a / $b;
}

try {
    echo divide(10, 0);  // Will throw an exception
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();  // Handles the exception
}
?>
```

Output: Error: Cannot divide by zero

10. How can you create a custom error handler in PHP?

To create a custom error handler in PHP, you can use the set_error_handler() function, which allows you to specify a custom function to handle errors.

Example:

```
<?php

// Custom error handler

function customErrorHandler($errno, $errstr) {

    echo "Custom Error: [$errno] $errstr<br>";

}


// Set the custom error handler

set_error_handler("customErrorHandler");


// Trigger an error

echo 10 / 0;

?>
```

Unit 11: File Handling in PHP

1. What is file handling in PHP?
File handling in PHP refers to the process of working with files (text files, images, etc.) on the server. It includes tasks like opening, reading, writing, and closing files, as well as manipulating file data such as creating, deleting, and renaming files.

File handling is important for applications that need to store data in a persistent way or interact with external files, such as logs, user uploads, or configuration files.

---

2. How do you open a file in PHP?
In PHP, you can open a file using the fopen() function. The fopen() function takes two main parameters: the file name (path) and the mode in which the file should be opened.

Syntax:

```
$file = fopen("file.txt", "r");  // Open file for reading
```

Modes:

- "r": Open the file for reading. The file must exist.
- "w": Open the file for writing. If the file doesn’t exist, it will be created. If it exists, it will be overwritten.

- "a": Open the file for appending. If the file doesn't exist, it will be created.
- "x": Open the file for writing only if it doesn't already exist.
- "r+": Open the file for reading and writing.
- "w+": Open the file for reading and writing, and overwrite it if it exists.

---

3. How do you read a file in PHP?
You can read a file in PHP using several functions depending on your needs. Common functions include:

- **fread()**: Reads a file in a binary-safe way.
- **fgets()**: Reads a line from the file.
- **file_get_contents()**: Reads the entire file into a string.

Example using fgets():

```
<?php

$file = fopen("file.txt", "r");

if ($file) {

    while (($line = fgets($file)) !== false) {

        echo $line . "<br>";
    }
    fclose($file);
} else {
    echo "Error opening the file!";
}
?>
```

Example using file_get_contents():

```
<?php
$content = file_get_contents("file.txt");
echo $content;
?>
```

---

4. How do you write data to a file in PHP?
You can write data to a file using functions like fwrite(), file_put_contents(), or fputs().

- **fwrite()**: Writes data to a file.
- **file_put_contents()**: Writes data to a file (overwrites existing content).

Example using fwrite():

```
<?php
```

```
$file = fopen("file.txt", "w");
if ($file) {
    fwrite($file, "Hello, World!");
    fclose($file);
} else {
    echo "Error opening the file!";
}
?>
```

Example using file_put_contents():

```
<?php
file_put_contents("file.txt", "Hello, World!");
?>
```

---

5. How do you check if a file exists in PHP?
You can use the file_exists() function to check if a file exists at a specific path.

Syntax:

```
if (file_exists("file.txt")) {
    echo "File exists!";
} else {
    echo "File does not exist.";
}
```

---

6. How do you delete a file in PHP?
You can delete a file using the unlink() function.

Syntax:

```
if (unlink("file.txt")) {
    echo "File deleted successfully.";
} else {
    echo "Error deleting the file.";
}
```

---

7. How do you rename a file in PHP?
You can rename a file using the rename() function.

Syntax:

```
if (rename("oldfile.txt", "newfile.txt")) {
    echo "File renamed successfully.";
} else {
    echo "Error renaming the file.";
}
```

---

8. How do you close a file in PHP?
After completing file operations, you should close the file using the fclose() function to free up resources.

Syntax:

```
fclose($file);
```

---

9. How can you append data to a file in PHP?
To append data to a file, you can use the "a" mode in fopen() or use file_put_contents() with the FILE_APPEND flag.

Example using fopen() in append mode:

```
<?php
$file = fopen("file.txt", "a");
if ($file) {
    fwrite($file, "Appended data.\n");
    fclose($file);
} else {
    echo "Error opening the file!";
}
?>
```

Example using file_put_contents() with FILE_APPEND:

```
<?php
file_put_contents("file.txt", "Appended data.\n", FILE_APPEND);
?>
```

---

10. How do you get the size of a file in PHP?
You can get the size of a file using the filesize() function.

Syntax:

```
$size = filesize("file.txt");
echo "File size: " . $size . " bytes";
```

---

11. How do you read a file line by line in PHP?
You can use the fgets() function to read a file line by line.

Example:

```
<?php
$file = fopen("file.txt", "r");
if ($file) {
    while (($line = fgets($file)) !== false) {
        echo $line . "<br>";
    }
    fclose($file);
} else {
    echo "Error opening the file!";
}
?>
```

---

12. How do you handle errors while working with files in PHP?

When working with files, it's essential to handle errors to avoid issues like trying to open a non-existent file or encountering permission problems. You can use error handling techniques such as try-catch blocks or checking file functions' return values.

Example using try-catch:

```
<?php
try {
    $file = fopen("nonexistentfile.txt", "r");
    if (!$file) {
        throw new Exception("File not found!");
    }
    fclose($file);
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
}
?>
```

Example using error checking:

```
<?php
$file = fopen("file.txt", "r");
if ($file) {
    // File operations
    fclose($file);
} else {
    echo "Error opening the file!";
}

?>
```

Unit 12: MySQL Database Handling in PHP

---

1. What is MySQL Database Handling in PHP?
MySQL Database Handling in PHP refers to the interaction between PHP and MySQL databases, enabling you to store, retrieve, update, and delete data in a MySQL database using PHP code. This is commonly done with the help of PHP's MySQL functions, such as mysqli and PDO (PHP Data Objects).

---

2. How do you connect to a MySQL database in PHP?
To connect to a MySQL database in PHP, you can use either the mysqli extension or PDO. Here's how you can do it using both methods:

Using mysqli:

```
<?php

$servername = "localhost";
```

```
$username = "root";

$password = "";

$dbname = "my_database";

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

 if ($conn->connect_error) {

    die("Connection failed: " . $conn->connect_error);

}

echo "Connected successfully";

?>
```

Using PDO:

```
<?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "my_database";

 try {

    $conn = new
PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);
```

```
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

    echo "Connected successfully";

}

catch(PDOException $e) {

    echo "Connection failed: " . $e->getMessage();

}

?>
```

---

3. How do you select data from a MySQL database in PHP?
To select data from a MySQL database, you can use SQL SELECT queries. After establishing a connection, you can execute a query to retrieve data from a table.

Using mysqli:

```
<?php

$conn = new mysqli("localhost", "root", "",
"my_database");

 $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"] . " - Email: " . $row["email"] . "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

Using PDO:

```
<?php
$conn = new
PDO("mysql:host=localhost;dbname=my_database",
"root", "");
 $sql = "SELECT id, name, email FROM users";
$stmt = $conn->prepare($sql);
$stmt->execute();
 while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "id: " . $row["id"] . " - Name: " . $row["name"]
. " - Email: " . $row["email"] . "<br>";
}
```

```
 $conn = null;
?>
```

---

4. How do you insert data into a MySQL database in PHP?

You can insert data into a MySQL database using the INSERT INTO SQL statement.

Using mysqli:

```
<?php
$conn = new mysqli("localhost", "root", "", "my_database");
 $name = "John Doe";
$email = "john@example.com";
$sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')";
 if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
```

```
$conn->close();
?>
```

Using PDO:

```
<?php
$conn = new
PDO("mysql:host=localhost;dbname=my_database",
"root", "");
$name = "John Doe";
$email = "john@example.com";
 $sql = "INSERT INTO users (name, email) VALUES
(:name, :email)";
$stmt = $conn->prepare($sql);
$stmt->bindParam(':name', $name);
$stmt->bindParam(':email', $email);
$stmt->execute();
echo "New record created successfully";
 $conn = null;
?>
```

5. How do you update data in a MySQL database in PHP?

To update data, you can use the UPDATE SQL query along with the SET clause.

Using mysqli:

```
<?php

$conn = new mysqli("localhost", "root", "",
"my_database");

$id = 1;

$name = "John Updated";

$email = "johnupdated@example.com";

$sql = "UPDATE users SET name='$name',
email='$email' WHERE id=$id";

 if ($conn->query($sql) === TRUE) {

    echo "Record updated successfully";

} else {

    echo "Error: " . $sql . "<br>" . $conn->error;

}

$conn->close();

?>
```

Using PDO:

```
<?php

$conn = new
PDO("mysql:host=localhost;dbname=my_database",
"root", "");

$id = 1;

$name = "John Updated";

$email = "johnupdated@example.com";

 $sql = "UPDATE users SET name=:name, email=:email
WHERE id=:id";

$stmt = $conn->prepare($sql);

$stmt->bindParam(':name', $name);

$stmt->bindParam(':email', $email);

$stmt->bindParam(':id', $id);

$stmt->execute();

 echo "Record updated successfully";

 $conn = null;

?>
```

---

6. How do you delete data from a MySQL database in PHP?

To delete data, you can use the DELETE FROM SQL query.

Using mysqli:

```
<?php

$conn = new mysqli("localhost", "root", "",
"my_database");

 $id = 1;

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

 if ($conn->query($sql) === TRUE) {

    echo "Record deleted successfully";

} else {

    echo "Error: " . $sql . "<br>" . $conn->error;

}

 $conn->close();

?>
```

Using PDO:

```
<?php

$conn = new
PDO("mysql:host=localhost;dbname=my_database",
"root", "");
```

```
 $id = 1;

$sql = "DELETE FROM users WHERE id=:id";

$stmt = $conn->prepare($sql);

$stmt->bindParam(':id', $id);

$stmt->execute();

 echo "Record deleted successfully";

 $conn = null;

?>
```

---

7. How do you handle errors while working with MySQL databases in PHP?
Error handling can be done using try-catch blocks in PDO, or checking if a query executed successfully in mysqli.

Using PDO:

```
<?php

try {

    $conn = new PDO("mysql:host=localhost;dbname=my_database", "root", "");
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
     $sql = "SELECT * FROM users";
    $stmt = $conn->prepare($sql);
    $stmt->execute();
    $result = $stmt->fetchAll();
  if (!$result) {
        throw new Exception("No records found.");
    }
 foreach ($result as $row) {
        echo $row['name'] . "<br>";
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
?>
```

Using mysqli:

```
<?php
$conn = new mysqli("localhost", "root", "",
"my_database");
```

```
 if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
 $sql = "SELECT * FROM users";
$result = $conn->query($sql);
 if ($result === FALSE) {
    echo "Error: " . $conn->error;
} else {
    while ($row = $result->fetch_assoc()) {
        echo $row["name"] . "<br>";
    }
}
 $conn->close();
?>
```

---

8. What is prepared statements in PHP and why are they used?
Prepared statements are used to execute SQL queries more securely by separating the query structure from the data. This helps to prevent SQL injection attacks. It

involves preparing the SQL query first, binding values to placeholders, and then executing the query.

Example:

```
<?php

$conn = new
PDO("mysql:host=localhost;dbname=my_database",
"root", "");

$name = "John Doe";

$email = "john@example.com";

$sql = "INSERT INTO users (name, email) VALUES
(:name, :email)";

$stmt = $conn->prepare($sql);

$stmt->bindParam(':name', $name);

$stmt->bindParam(':email', $email);

$stmt->execute();

 echo "Record inserted successfully";

$conn = null;

?>
```

9. What is the difference between mysqli and PDO in PHP?

- **mysqli** (MySQL Improved): It is specific to MySQL databases and provides both a procedural and object-oriented approach. It supports prepared statements and transactions.
- **PDO** (PHP Data Objects): It is a database abstraction layer, meaning it can work with different types of databases (not just MySQL). It only supports object-oriented programming and also supports prepared statements.

Main Differences:

- **Database support**: mysqli only supports MySQL, while PDO supports multiple databases.
- **Syntax**: mysqli provides both procedural and object-oriented syntax, whereas PDO is strictly object-oriented.
- **Features**: Both support prepared statements, but PDO provides more flexibility for using different database types.
