- **What is PHP?**

  The PHP stands for Hypertext Preprocessor. It was abbreviated previously as Personal Home Page. PHP is a server-side scripting language designed for web development. It is widely used for creating dynamic web pages and can be embedded within HTML code. PHP is an open-source language, and it is executed on the server, generating HTML content that is then sent to the client's web browser for display.

  Key features and uses of PHP

  1. **Dynamic Web Pages:** PHP is widely used for creating dynamic web pages, allowing developers to embed PHP code within HTML for interactive content.
  2. **Server-Side Execution:** PHP code is executed on the server, generating HTML content sent to the client's web browser for display.
  3. **Open Source:** PHP is an open-source language, freely available for use and modification, contributing to a large and active community.
  4. **Database Integration:** PHP facilitates seamless interaction with databases, supporting tasks such as data retrieval, insertion, updating, and deletion.
  5. **Cross-Platform Compatibility:** PHP is platform-independent, capable of running on various operating systems like Windows, Linux, and macOS.
  6. **Extensibility:** Developers can enhance PHP's functionality through extensions, adding features and capabilities as needed.
  7. **Frameworks:** Supports frameworks for efficient application development.

→ **What is the difference between PHP and HTML.**

PHP (Hypertext Preprocessor): Used for server-side scripting, generating dynamic content, and interacting with databases.

HTML (Hypertext Markup Language): Primarily for structuring static web content on the client-side.

PHP (Hypertext Preprocessor):

- **Nature:** Dynamic (can change).
- **Execution:** Works on the web server (server-side).
- **File Extension:** ".php."
- **Usage:** Generates dynamic content and interacts with databases on the server.

HTML (Hypertext Markup Language):

- **Nature:** Static (doesn't change).
- **Execution:** Works on the user's device (client-side).
- **File Extension:** ".html" or ".htm."
- **Usage:** Defines the basic structure of web pages.

→ **Note:** Install Xampp for PHP running.
→ Go to C:\xampp\htdocs Create a folder PHPproject
→ To run all file: Open XAMPP control panel then Start Apache and MySQL
→ **localhost/foldername/**↲
http://localhost/PHPproject/
→ To run a specific file:
http://**localhost/PHPproject/class_1.php/**↲

Structure of PHP programme:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
    <?php  // PHP code embedded within the body of the HTML document
    echo "Hello PHP";
    ?>
</body>
</html>
```

`Data types in PHP:`

1. **Integers:**
   - Whole numbers without decimals.
     Example: $num = 42;

2. **Floats:**
   - Numbers with decimals.
     Example: $floatNum = 3.14;

3. **Strings:**
   - Sequences of characters.
     Example: $str = "Hello, PHP";

4. **Booleans:**
   - Represents true or false.
     Example: $isTrue = true;

5. **Arrays:**
   - Ordered map of values.
     Example: $colors = array("red", "green", "blue");

6. **Objects:**
   - Instances of user-defined classes.
     Example:
     ```
     class Car {
         public $color;
         public $model;
     }
     $myCar = new Car();
     ```

7. **NULL:**
   - Represents absence of a value.
     Example: $var = null;

8. **Resource:**
   - Holds a reference to an external resource.
     Example:
     ```
     // Example: $file is a resource returned by fopen()
        $file = fopen("example.txt", "r");
     ```

→ **Summation of two number:**

```
<body>
  <?php
  $x=11;
  $y=15;
  $add= $x + $y;
  echo "Sum of $x and $y= $add";
  ?>
</body>
```

→ **String functions:**

```
<body>
  <?php
  $str="Hello world";
  echo "Length of string ", strlen($str),"<br>" ;
```

```
    echo str_word_count("Hypertext Preprocessor"),"<br>";
    echo "Reverse of HELLO = ",strrev("HELLO"), "<br>";
    echo str_replace("LG","Samsung", "Hello LG");          //Output: Hello Samsung
    ?>
  </body>
```

→ **Operators in PHP:**

1. **Arithmetic Operators:**
   - **+, -, *, /, %**

2. **Comparison Operators:**
   - **==, !=** (or **<>**), **<, >, <=, >=**

3. **Logical Operators:**
   - **&&** (or **and**), **||** (or **or**), **!**

4. **Assignment Operators:**
   - **=, +=, -=, *=, /=, %=**

5. **Increment/Decrement Operators:**

   ++ (Increment by 1)

   -- (Decrement by 1)

   Example:
   ```
   $counter = 5;
   $counter++;      // Increment by 1 // $counter is now 6
   ```

6. **Concatenation Operator: . (dot)**

```
<body>
  <?php
   $txt1="Hello";
   $txt2="Banglore";
   $txt3=$txt1.$txt2;
   echo $txt3, "<br>";        //output: HelloBanglore
   $txt1.=$txt2;
   echo $txt1;               //output: HelloBanglore
  ?>
</body>
```

→ **Ternary conditional operator: (?:)**

```
  <?php
   $a=4;
   $b=6;
   $c=8;
   $val=($a>$b)? (($a>$c)? $a : $c) : (($b>$c)? $b : $c);
   echo $val;
  ?>
```

→ **What is the difference between == and === in PHP?**

1. **== (Equality Operator):**
   - Checks if values are the same, even if they're different types.
   - Doesn't care about the data types.
2. **=== (Identity Operator):**
   - Checks if values are exactly the same, including data types.

- Cares about both the value and the type.

- **Explain the MVC architecture and its importance in PHP applications.**
  MVC (Model-View-Controller) is a software architectural pattern widely used in web development, and it separates an application into three interconnected components: Model, View, and Controller.
  Model:
  Represents the application's data and business logic.
  Manages data storage, retrieval, and manipulation.
  Independent of the user interface.
  View:
  Represents the presentation and user interface.
  Displays data to the user and receives user input.
  Renders information from the Model to the user.
  Controller:
  Acts as an intermediary between the Model and View.
  Processes user input, manipulates data in the Model, and updates the View accordingly.
  Handles application logic and flow.

The MVC architecture is crucial in PHP applications for maintaining code quality, enhancing collaboration, and facilitating adaptable development. This approach leads to higher code quality and supports sustainable project growth over time.

Find greatest number.

```php
<?php
   $a=5;
   $b=7;
   if($a>5)
     {
      echo "$a is greater";
     }
    else if($a==$b)
     {
      echo "Both numbers are equal";
     }
    else
     {
      echo "$b is greater";
     }
  ?>
```

- **Conditional statements :** PHP supports several conditional statements that allow us to control the flow of our code based on different conditions.

  → **if statement:**
  ```
  if (condition) {
     // code to be executed if the condition is true
  }
  ```
  → **if-else statement:**
  ```
  if (condition) {
     // code to be executed if the condition is true
  } else {
     // code to be executed if the condition is false
  }
  ```
  Note: Executes one block of code if the condition is true and another block if the condition is false.

→ **if-elseif-else statement:**

```
if (condition1) {
   // code to be executed if condition1 is true
} elseif (condition2) {
   // code to be executed if condition2 is true
} else {
   // code to be executed if none of the conditions are true
}
```

→ **Switch statement:**

```
switch (variable) {
   case value1:
      // code to be executed if variable equals value1
      break;
   case value2:
      // code to be executed if variable equals value2
      break;
   // additional cases...
   default:
      // code to be executed if none of the cases match
}
```

→ **ternary conditional operator:** It provides a shorthand way to write simple if-else statements in a single line.

```
$result = (condition) ? true_expression : false_expression;
```

→ **null coalescing operator:**

```
$result = $variable ?? $default_value;
```

Note: Returns the value of $variable if it is set and not null, otherwise, it returns the $default_value.

- **Loops:** PHP supports several types of loops that allow us to repeatedly execute a block of code.

→ **while loop:** Executes a block of code repeatedly as long as the specified condition is true. It checks the condition before each iteration.

```
while (condition) {
   // code to be executed as long as the condition is true
}
```

→ **do-while loop:** Similar to the while loop, but it checks the condition after each iteration. This ensures that the code inside the loop is executed at least once.

```
do {
   // code to be executed at least once
} while (condition);
```

→ **for loop:** Executes a block of code repeatedly as long as the specified condition is true. It typically includes an initialization step, a condition to check before each iteration, and an increment or decrement step.

```
for (initialization; condition; increment/decrement) {
   // code to be executed in each iteration
}
```

→ **foreach loop:** Iterates over each element in an array, assigning the current element's value to the variable specified ($value in this case).

```
foreach ($array as $value) {
   // code to be executed for each element in the array
}
```

→ **break statement / continue statement::**

```
while (condition) {
   // code
   if (some_condition) {
      continue; // skip the rest of the code in this iteration and move to the next      OR
      break; // exit the loop prematurely
   }
   // more code
}
```

→ **Table of a number:**

```
<body>
 <?php
   for($i=1; $i<=10; $i++){
    $t=$i*7;
    echo "7 X $i = $t <br>";
   }
  ?>
</body>
```

→ **Even numbers between 1 to 100:**

```
<body>
 <?php
 echo "Even numbers are following: <br>";
   $i=1;
   while($i<=100){
    if($i%2==0){
      echo "$i, ";
    }
    $i++;
   }
  ?>
</body>
```

→ **Array implementation with foreach loop:**

```
<body>
 <?php
   $Arr=array("Bengalore","Kolkata","Bhuneshwar",1,2,3);
   foreach($Arr as $value){
    echo "$value, <br>";
   }
   echo "Current date is: ".date("d/m/y");
  ?>
</body>
```

```
echo "Root of: ".sqrt(16);
```

→ **Function:**

```
<body>
  <?php
  function addition($x=1, $y=7)
        {           //Function takes two parameters $x and $y with default values of 1 and 7, respectively.
          $sum=$x + $y;
          return $sum;
        }
      $value=addition(6,8);
      echo "Addition: ",$value;


  // Define an anonymous function and assign it to the variable $multiplication
      $multiplication = function($a, $b)
        {
          $mul = $a * $b;
          echo "Multiplication: ", $mul;
        };
      $multiplication(7, 8);
    ?>
</body>
```

→ **In PHP, an array** is a data structure that allows us to store multiple values in a single variable. PHP supports several types of arrays, each with its own characteristics and use cases.

1. **Indexed Arrays:** The most basic type of array in PHP. Elements are stored with numeric indices starting from 0. Example:

```
<?php
    $colors = array("Red", "Green", "Blue");
    print_r($colors); //Output: Array ( [0] => Red [1] => Green [2] => Blue )
    echo $colors[0]; //Output: Red
 ?>
```

2. **Associative Arrays:** Elements are stored with named keys instead of numeric indices. Useful when we want to associate values with specific names. Example:

```
<?php
    $person = array("name" => "John", "age" => 30, "city" => "New York");
    print_r($person);    //Output: Array ( [name] => John [age] => 30 [city] => New York )
 ?>
```

3. **Multidimensional Arrays:** Arrays containing one or more arrays. Useful for representing tables or matrices. Example:

```
<?php
    $matrix = array(
      array(1, 2, 3),
      array(4, 5, 6),
      array(7, 8, 9)
  );
  print_r($matrix);
?>  //Output:  Array ( [0] => Array ( [0] => 1 [1] => 2 [2] => 3 ) [1] => Array ( [0] => 4 [1] => 5 [2] => 6 ) [2] => Array ( [0] => 7 [1] => 8 [2] => 9 ) )
```

4. **Short Array Syntax (PHP 5.4 and later):** A shorthand syntax to define arrays. Example:

```
<?php
    $colors = ["Red", "Green", "Blue"];
    print_r($colors);  //Output: Array ( [0] => Red [1] => Green [2] => Blue )
?>
```

5. **List:** Used to assign values to a list of variables in a single operation. Often used in conjunction with the **list()** function. Example:

```
<?php
    list($first, $second, $third) = array("Apple", "Banana", "Orange");
    echo $first,$second,$third;
 ?>
```

6. **Compact:** Creates an array from variables and their values. Example:

```
<?php
     $name = "John";
     $age = 30;
     $city = "New York";
     $person = compact("name", "age", "city");
     print_r($person);
?>          //Output: Array ( [name] => John [age] => 30 [city] => New York )
```

7. **Range:** Generates an array containing a range of elements. Example:

```
<?php
   $numbers = range(1, 5);
   print_r($numbers);
?>    //Output: Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 )
```

→ **Array implementation 1D:**

```
<body>
   <?php
     $phones=array("LAVA","SAMSUNG","LG","SONY","APPLE");
     $arrLen=count($phones);
     for($i=0; $i<$arrLen; $i++)
       {
        echo $phones[$i],",";
       }
   ?>    //Output: LAVA,SAMSUNG,LG,SONY,APPLE,
</body>
```

→ **Array implementation 2D:**

```
<?php
  $accessories=array(
    array("SONY","LAVA","SAMSUNG"),
    array("HP","DELL"),
    array("Bharat","Rusia","Amercia","England")
  );

  for($i=0; $i<count($accessories); $i++)
    {
      for($j=0; $j<count($accessories[$i]); $j++)
        {
          echo $accessories[$i][$j],", ";
        }
        echo "<br>";
    }
 ?>
```

Core java:

```
class JaggedArray {
   public static void main(String[] args) {
      int Arr[][]={{1,2,3,4}, {2,3,4,5},{ 1,2,3,},{1,5}};
      for(int i=0; i<Arr.length; i++){
         for(int j=0; j<Arr[i].length; j++) {
            System.out.print(Arr[i][j]+", ");
         }
         System.out.println();
      }
   }
}
```

Output:

```
SONY, LAVA, SAMSUNG,
HP, DELL,
Bharat, Rusia, Amercia, England,
```

→ In PHP, **$_GET** and **$_POST** are superglobal arrays that allow us to collect form data after submitting an HTML form using the GET or POST method, respectively. These methods are commonly used to send data from the client (browser) to the server.

→ **$_GET Method:** Used to collect data sent in the URL parameters. It appends the data to the URL.
Security: Less secure than $_POST because the data is visible in the URL.
Usage: Suitable for non-sensitive data and when we want to share data via a link.
Example:

```
<body>
      <form action="process.php" method="get">
          Name: <input type="text" name="username">
          <input type="submit" value="Submit">
      </form>
</body>
```

Note: Here PHP file name must be **process.php** such as written within html form action mode.

```
<?php
 if(isset($_GET['username'])) {
     $username = $_GET['username'];
     echo "Hello, $username!";
 }
?>           //Output: Run html page, then input the data in input box and click
on submit button.
```

→ **$_POST Method:** Used to collect data sent in the HTTP request body. It does not append data to the URL.
Security: More secure than **$_GET** because the data is not visible in the URL.
Usage: Suitable for sensitive data like passwords and when we don't want data to be visible in the URL.

```
<!-- HTML form using POST method -->
<form action="process.php" method="post">
    Password: <input type="password" name="password">
    <input type="submit" value="Submit">
</form>
```

// Note: Here PHP file name must be **process.php** such as written within html form action mode.

```
<?php
if(isset($_POST['password'])) {   // If set, assign the value to the $password variable
    $password = $_POST['password'];
    echo "Password submitted securely!";
}
?>
```

Difference between GET and POST method:

GET:	POST:
-Sends data through the URL. -Less secure for sensitive information. -Limited data size. -Can be cached, visible in browser history. -Suitable for retrieving data.	-Sends data in the request body. -More secure for sensitive information. -Can handle larger data. -Not cached, not visible in browser history. -Suitable for submitting or modifying data.

- **Form validation:** PHP form validation is a process of ensuring that the data submitted through a web form meets certain criteria or constraints before it is processed or stored. The purpose of form validation is to enhance the security and accuracy of the data collected from users. It helps prevent invalid or malicious data from being submitted, improving the overall integrity of the application.

  HTML: <body>

```
  <form action="process.php" method="post">
   <label for="name">Name:</label>
   <input type="text" id="name" name="name" >
   <input type="submit" value="Submit">
</form>


<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {   // Check if the form is submitted using the POST method.
   $name = $_POST["name"];          // Collect the value of the "name" field from the form

   if (empty($name)) {     // Validate the "name" field
      echo "Name is required.";
   } else {
      echo "Hello, $name!";       // If the "name" field is not empty, process the valid data
   }
}
?>
```

→ Example2 of PHP form validation:

```
<?php
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
       $name = $_POST["name"];
       $address = $_POST["address"];

       if (empty($name)) {
          echo "Name is required.";
       } else {
          echo "Hello, $name!";
       }

       if (empty($address)) {
          echo "Address is required.";
       }

       if (empty($name) || empty($address)) {  // Check if both name and address are not empty
          echo "Enter required fields";
       } else {
          echo "Form submitted successfully";
          // Additional processing or actions can be performed here
       }
    }
?>
```

→ Example3 of PHP form validation:

```
<?php
    if ($_SERVER["REQUEST_METHOD"] == "POST") {     // Check if form is submitted
         $name = $_POST["name"];        // Collect form data
         $email = $_POST["email"];

       $errors = array();            // Initialize an array to store validation errors

       if (empty($name)) {       // Validate name
          $errors[] = "Name is required.";
       }
       if (empty($email)) {        // Validate email
          $errors[] = "Email is required.";
       } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $errors[] = "Invalid email format.";
        }

        if (empty($errors)) {         // If no validation errors, process the data
            echo "Form submitted successfully!<br>";
            // Additional processing or database insertion can be done here
        } else {  // Display validation errors
            foreach ($errors as $error) {
                echo $error . "<br>";
            }
        }
    }
?>
```

- **PHP with MySQL:** PHP and MySQL are commonly used together to create dynamic web applications. PHP is a server-side scripting language, while MySQL is a relational database management system.

→ **Connecting to MySQL Database:** PHP provides functions to connect to a MySQL database using the **mysqli** extension or **PDO** (PHP Data Objects). We need to provide the database credentials (hostname, username, password, and database name).

Example:

```
<?php        // Database connection parameters
 $servername = "localhost";   // Database server name (usually 'localhost' for local development)
 $username = "root";       // Database username
 $password = "";         // Database password
 $dbname = "example_database";       // Database name

 // Create a connection to the MySQL database
 $conn = new mysqli($servername, $username, $password, $dbname);

 // Check if the connection was successful
 if ($conn->connect_error) {         // If connection failed, terminate the script and display an error message
    die("Connection failed: " . $conn->connect_error);
 }                                   // If connection is successful, we can proceed with database operations
?>
```

→ **Executing MySQL Queries:** We can use PHP to execute SQL queries and interact with the MySQL database. The example below inserts data into a table.

```
<?php
 $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;
 }
?>
```

→ **Fetching Data from MySQL:** PHP can retrieve data from MySQL using SELECT queries. The example below retrieves and displays user data from the "users" table.

```
<?php        // Execute a SELECT query to retrieve all columns from the "users" table
    $result = $conn->query("SELECT * FROM users");

    if ($result->num_rows > 0) {     // Check if there are any rows in the result
       while ($row = $result->fetch_assoc()) {       // Loop through each row in the result set
          // Display user data, assuming the "users" table has columns named "name" and "email"
          echo "Name: " . $row["name"] . " - Email: " . $row["email"] . "<br>";
       }
```

```
   } else { // If there are no results, echo a message indicating so
      echo "0 results";
   }
?>
```

→ **Closing the Database Connection:** It's good practice to close the MySQL database connection when it's no longer needed.

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

❖ **Example of using** PHP with MySQL to create a database, create an employee table, and perform basic operations such as inserting, updating, deleting, and fetching data:

→ **Create a Database and Employee Table:**

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

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

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

 $sqlCreateDB = "CREATE DATABASE IF NOT EXISTS company";     // Create database
 if ($conn->query($sqlCreateDB) === TRUE) {
    echo "Database created successfully.<br>";
 } else {
    echo "Error creating database: " . $conn->error . "<br>";
 }

 $conn->select_db("company");     // Select the database

 $sqlCreateTable = "CREATE TABLE IF NOT EXISTS employees (                // Create employee table
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    position VARCHAR(50) NOT NULL,
    salary INT(10) NOT NULL
 )";

 if ($conn->query($sqlCreateTable) === TRUE) {
    echo "Employee table created successfully.<br>";
 } else {
    echo "Error creating table: " . $conn->error . "<br>";
 }

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

→ **Insert Data into Employee Table:**

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

 // Insert data into the employee table
 $sqlInsert = "INSERT INTO employees (name, position, salary) VALUES ('John Doe', 'Developer', 50000)";
 if ($conn->query($sqlInsert) === TRUE) {
    echo "Record inserted successfully.<br>";
 } else {
    echo "Error inserting record: " . $conn->error . "<br>";
 }

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

→ **Update Data in Employee Table:**

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

 // Update data in the employee table
 $sqlUpdate = "UPDATE employees SET salary = 55000 WHERE name = 'John Doe'";
 if ($conn->query($sqlUpdate) === TRUE) {
    echo "Record updated successfully.<br>";
 } else {
    echo "Error updating record: " . $conn->error . "<br>";
 }

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

→ **Delete Data from Employee Table:**

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

 $sqlDelete = "DELETE FROM employees WHERE name = 'John Doe'";     // Delete data from the employee table
 if ($conn->query($sqlDelete) === TRUE) {
    echo "Record deleted successfully.<br>";
 } else {
    echo "Error deleting record: " . $conn->error . "<br>";
 }

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

→ **Fetch Data from Employee Table:**

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

 $result = $conn->query("SELECT * FROM employees");   // Fetch data from the employee table

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

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

- **Note: MySQL database connection in PHP.**
  Using new mysqli: $conn = new mysqli("localhost", "root", "", "company");
  This approach uses the object-oriented style to create a new instance of the **mysqli** class. It provides a more modern and object-oriented way of interacting with MySQL databases. It allows for more fine-grained control and access to additional features provided by the **mysqli** class.

  Using mysqli_connect: $conn = mysqli_connect("localhost", "root", "", "company");
  This approach uses the procedural style and the **mysqli_connect** function to establish a connection. The procedural style is simpler and might be more familiar to users coming from older versions of PHP or other programming languages. However, it may lack some of the features and flexibility offered by the object-oriented approach.

- **PHP with MySQL using procedural style:**
  Page1: Database creation:

```
<?php
    $servername = "localhost";        // Database connection parameters
    $username = "root";
    $password = "";

    // Establish a connection to the MySQL server
    $conn = mysqli_connect($servername, $username, $password);
    if (!$conn) {        // Check if the connection was successful
       // If connection fails, terminate the script and display an error message
       die("Connection failed: " . mysqli_connect_error());
    } else {
       echo "Connected successfully";       // If connected successfully, display a success message
    }

    $sql = "CREATE DATABASE Employee";      // Creating a database named "Employee"

    if (mysqli_query($conn, $sql)) {  // Execute the SQL query to create the database
          echo "Database created successfully";   // If the query is successful, display a success
    message
    } else {
       // If there's an error during database creation, display an error message
       echo "Error! During creating database: " . mysqli_error($conn);
    }

    mysqli_close($conn);   // Close the database connection
?>
```

Page2: Table creation:

```
<?php
    $servername = "localhost";      // Database connection parameters
    $username = "root";
    $password = "";
    $dbname = "Employee";

    // Establish a connection to the MySQL server with specified database
    $conn = mysqli_connect($servername, $username, $password, $dbname);

    if (!$conn) {  // Check if the connection was successful
       // If connection fails, terminate the script and display an error message
       die("Error to database connection: " . mysqli_connect_error());
    }

    // SQL query to create a table named "Employee_data" in the specified database
    $sql = "CREATE TABLE Employee_data (
       emp_id INT PRIMARY KEY,
       emp_name VARCHAR(25) NOT NULL,
       emp_address VARCHAR(55),
       emp_mobile VARCHAR(12) NOT NULL,
       emp_email VARCHAR(25),
       job_title VARCHAR(25) NOT NULL
    )";

    if (mysqli_query($conn, $sql)) {    // Execute the SQL query to create the table
       // If the query is successful, display a success message
       echo "Employee_data table created successfully";
    } else {
       // If there's an error during table creation, display an error message
```

```
            echo "Error! " . mysqli_error($conn);
        }

        // Close the database connection
        mysqli_close($conn);
    ?>
```

Page3: Data insertion:

```
<?php
        $servername = "localhost";    // Database connection parameters
        $username = "root";
        $password = "";
        $dbname = "Employee";

        // Establish a connection to the MySQL server with the specified database
        $conn = mysqli_connect($servername, $username, $password, $dbname);

        if (!$conn) {
            // If connection fails, terminate the script and display an error message
            die("Error to database connection: " . mysqli_connect_error());
        }

        // SQL query to insert a new record into the "Employee_data" table
        $sql = "INSERT INTO Employee_data (emp_id, emp_name, emp_address, emp_mobile, emp_email,
        job_title) VALUES  (111, 'Anup', 'Bangalore', '8978675645', 'anp@gmail.com', 'SE')";

        if (mysqli_query($conn, $sql)) {    // Execute the SQL query to insert the record
            echo "A new record inserted successfully";       // If the query is successful, display a success message
        } else {
            // If there's an error during insertion, display an error message
            echo "Error! " . mysqli_error($conn);
        }

        mysqli_close($conn);    // Close the database connection
?>
```

Page4: Data view

```
<?php
        $servername = "localhost";    // Database connection parameters
        $username = "root";
        $password = "";
        $dbname = "Employee";

        // Establish a connection to the MySQL server with the specified database
        $conn = mysqli_connect($servername, $username, $password, $dbname);

        if (!$conn) {     // Check if the connection was successful
            // If connection fails, terminate the script and display an error message
            die("Error to database connection: " . mysqli_connect_error());
        }

        // SQL query to select data from the "Employee_data" table
        $sql = "SELECT * FROM Employee_data WHERE emp_id=111";      // OR $sql = "SELECT * FROM
        Employee_data";

        $result = mysqli_query($conn, $sql);       // Execute the SQL query

        if (mysqli_num_rows($result) > 0) {    // Check if there are any rows in the result
           // Loop through each row in the result set
           while ($row = mysqli_fetch_assoc($result)) {   // Display the selected data for each row
              echo "ID: " . $row["emp_id"] . " Name: " . $row["emp_name"] . " Address: " .
        $row["emp_address"] . " Mobile: " . $row["emp_mobile"] . " Email: " . $row["emp_email"] . "
        Job_title: " . $row["job_title"] . "<br>";
           }
        } else {  // If there are no results, echo a message indicating so
              echo "No record found";
        }
        mysqli_close($conn);     // Close the database connection
?>
```

Page5: Data updating

```
<?php
        $servername = "localhost";   // Database connection parameters
        $username = "root";
        $password = "";
        $dbname = "Employee";

        // Establish a connection to the MySQL server with the specified database
        $conn = mysqli_connect($servername, $username, $password, $dbname);

        if (!$conn) {  // Check if the connection was successful
           // If connection fails, terminate the script and display an error message
           die("Error to database connection: " . mysqli_connect_error());
        }

        // SQL query to update the "emp_name" field for the record with "emp_id" equal to 111
        $sql = "UPDATE Employee_data SET emp_name='Anuj' WHERE emp_id=111";

        if (mysqli_query($conn, $sql)) {    // Execute the SQL query to update the record
           echo "Record updated successfully!";   // If the query is successful, display a success message
        } else {
           // If there's an error during the update, display an error message
           echo "Error! " . mysqli_error($conn);
        }

        mysqli_close($conn);   // Close the database connection
?>
```

Page6: Data deleting

```
        <?php
                $servername = "localhost";   // Database connection parameters
                $username = "root";
                $password = "";
                $dbname = "Employee";
        // Establish a connection to the MySQL server with the specified database
        $conn = mysqli_connect($servername, $username, $password, $dbname);

        if (!$conn) {  // Check if the connection was successful
           // If connection fails, terminate the script and display an error message
           die("Error to database connection: " . mysqli_connect_error());
        }

        $sql = "DELETE FROM Employee_data WHERE emp_id=112";

        if (mysqli_query($conn, $sql)) {    // Execute the SQL query to update the record
           echo "Record Deleted successfully!";   // If the query is successful, display a success message
        } else {
           // If there's an error during the update, display an error message
           echo "Error! " . mysqli_error($conn);
        }

        mysqli_close($conn);   // Close the database connection
    ?>
```

- How do you connect html form with mysql using php?

  To connect an HTML form with MySQL using PHP, we need to follow these steps:

  1. **Create a Database:**
     - Make sure you have a MySQL database set up.
     - Create a table to store the form data.
  2. **Create an HTML Form:**
     - Design an HTML form to collect data from users.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form to MySQL</title>
</head>
<body>
  <form action="process.php" method="post">
    <label for="name">Name:</label>
    <input type="text" name="name" id="name" required>

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

    <button type="submit">Submit</button>
  </form>
</body>
</html>
```

Create a PHP Script to Process Form Data:

- Create a PHP script (**process.php**) to handle the form submission.
- Retrieve data from the form using the **$_POST** superglobal.
- Establish a connection to the MySQL database.
- Insert the form data into the database.

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

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

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

        // Retrieve form data
        $name = $_POST['name'];
        $email = $_POST['email'];

        $sql = "INSERT INTO your_table_name (name, email) VALUES ('$name', '$email')";  // Insert data into the database


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

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

- **How do you upload files in PHP?**
  To upload files in PHP, we can use the **$_FILES** superglobal along with an HTML form that has the attribute **enctype="multipart/form-data"**.

  Create an HTML form with the enctype attribute set to "multipart/form-data" to allow file uploads.

```
<form action="upload.php" method="post" enctype="multipart/form-data">
  <input type="file" name="fileInput">
  <button type="submit" name="submit">Upload</button>
</form>
```

  Create a PHP Script to Handle File Upload (upload.php):
  - Retrieve the uploaded file using the **$_FILES** superglobal.
  - Check for errors during the upload process.
  - Move the uploaded file to a desired location on the server.

```
<?php
    if ($_FILES['fileInput']['error'] == UPLOAD_ERR_OK) {  // Check if a file was submitted
      $targetDir = "uploads/"; // Specify the directory where you want to store the uploaded files
      $targetFile = $targetDir . basename($_FILES['fileInput']['name']);

      // Move the file to the specified directory
      if (move_uploaded_file($_FILES['fileInput']['tmp_name'], $targetFile)) {
        echo "The file " . basename($_FILES['fileInput']['name']) . " has been uploaded.";
      } else {
        echo "Sorry, there was an error uploading your file.";
      }
    } else {
        echo "Error: " . $_FILES['fileInput']['error'];
    }
?>
```

Inheritance:
```
class Animal {
    public function eat() {
        echo "Animal is eating.";
    }
}
class Dog extends Animal {
    public function bark() {
        echo "Dog is barking.";
    }
}

$dog = new Dog();
$dog->eat();      // Inherited method
$dog->bark();     // Class-specific method
```

- **What is the purpose of sessions and cookies in PHP?**
  Sessions: Sessions are a way to persist data across multiple requests made by the same user during their visit to a website.
  Usage: Store user-specific information, maintain user state between pages.
  Cookies: Cookies are small pieces of data stored on the client's browser, which can be sent back to the server with subsequent requests.
  Usage: Store user preferences, track user activity.

- **What is the purpose of Composer in PHP?**
  Composer is a dependency manager for PHP, designed to facilitate the management of libraries and packages in PHP projects. Its main purposes include:
  - Manages project dependencies.
  - Generates autoloader for class loading.
  - Simplifies project setup and configuration.
  - Facilitates version management.

  Usage: Declared in **composer.json**, operated through command line.
  Benefits: Saves time, ensures consistency, promotes modular code.

|| ॐ एकदन्ताय विद्महे वक्रतुण्डाय धीमहि तन्नो दन्तिः प्रचोदयात् ||