PHP IN ONE GLANCE

THE BASIC AND BEYOND

[Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.]

Contents

Class 1

Introduction to Server-Side Scripting with PHP

Objective:

Introduce students to server-side scripting, explain its importance in web development, and provide a basic understanding of how PHP is used to process data on the server before sending it to the client. By the end, students should understand the role of server-side scripts and how to set up a PHP environment.

Definition:

Server-side scripting is a method of running scripts on the web server rather than the user's browser (client-side). PHP is one of the most popular server-side scripting languages used for creating dynamic web pages.

- **Explanation:**
  - **Client-Side vs. Server-Side**: Client-side code (HTML, CSS, JavaScript) runs on the user's browser, while server-side code (PHP, Python, etc.) runs on the web server.
  - PHP processes code on the server and generates HTML, which is then sent to the user's browser.

Setting Up the PHP Development Environment

To run PHP code, we need a server environment. The easiest way to set this up locally is by using software like **XAMPP**, **WAMP**, or **MAMP**.

Why XAMPP, WAMP, or MAMP?

These tools bundle the Apache server, MySQL (database), and PHP (language interpreter) together, creating a simple, one-click solution to set up a local server on your computer.

Steps to Install XAMPP (for Windows)

1. **Download XAMPP:**
   - Go to the XAMPP website.
   - Choose the XAMPP installer for your operating system (usually Windows, macOS, or Linux).
2. **Install XAMPP:**
   - Open the downloaded file and follow the on-screen instructions to install.
   - Choose components to install (select Apache, MySQL, and PHP by default).
   - Select the installation directory (default is C:\xampp on Windows).
3. **Start Apache and MySQL:**
   - Open the XAMPP Control Panel.

- o Click “Start” for Apache and MySQL to launch the web server and database server.
- o Ensure both services are running by checking for green indicators.

Using XAMPP on Other Operating Systems:

- **macOS:** Use MAMP, which works similarly to XAMPP but is optimized for macOS.
- **Linux:** Use XAMPP or LAMP (Linux + Apache + MySQL + PHP).

---

4. Testing the PHP Installation

Once XAMPP is installed and running, you can test your PHP setup by creating a simple PHP script.

Steps to Create and Run a PHP Script

1. **Locate the htdocs Folder:**
   - o Navigate to the htdocs folder inside your XAMPP directory (usually C:\xampp\htdocs on Windows).
   - o This is the default folder where all your PHP files should be saved to run in the browser.
2. **Create a New PHP File:**
   - o Inside the htdocs folder, create a new file named test.php.

   - Open this file in a text editor and add the following code:

```
<?php
// This script displays information about the PHP installation.
phpinfo();
?>
```

   - The phpinfo() function outputs information about PHP’s configuration, which is useful to verify that PHP is working correctly.

3. **Run the Script:**
   - Open your web browser.
   - Type http://localhost/test.php in the address bar and press Enter.
   - If PHP is installed correctly, you should see a page displaying details about your PHP configuration.

- **Example**:

```
<?php
   echo "Hello from the server!";
?>
```

Assignments

1. Write a PHP script that displays "Hello, World!".
2. Create a script to display today’s date and time using PHP.
3. Create a webpage using HTML and PHP to show a welcome message that changes based on the current time.
4. Write a PHP script to display your name and age using PHP.
5. Explain in your own words the difference between client-side and server-side scripting.

SOLUTION

1. PHP Script to Display "Hello, World!"

```
<?php

// Displaying a simple message

echo "Hello, World!";

?>
```

2. Script to Display Today’s Date and Time in PHP

```
<?php

// Displaying today's date and time

echo "Today's date and time is: " . date("Y-m-d H:i:s");

?>
```

3. Webpage Showing a Time-Based Welcome Message

HTML with Embedded PHP:

```
<!DOCTYPE html>
<html>
<head>
  <title>Time-Based Welcome Message</title>
</head>
<body>
  <?php
      date_default_timezone_set("Your/Timezone"); // Replace "Your/Timezone" with your timezone, e.g., "Asia/Kolkata"
  $hour = date("H");
  if ($hour < 12) {
     echo "<h1>Good Morning!</h1>";
  } elseif ($hour < 18) {
     echo "<h1>Good Afternoon!</h1>";
  } else {
     echo "<h1>Good Evening!</h1>";
  }
  ?>
</body>
```

```
</html>
```

4. PHP Script to Display Your Name and Age

```
<?php
$name = "John Doe"; // Replace with your name
$age = 25; // Replace with your age
 echo "My name is $name and I am $age years old.";
?>
```

5. Difference Between Client-Side and Server-Side Scripting

Client-Side Scripting

- **Definition**: Executes on the user's web browser. Scripts are downloaded with the webpage and run on the client's machine.
- **Languages Used**: JavaScript, HTML, CSS.
- **Key Features**:
    - Provides interactivity and visual effects (e.g., animations, form validations).
    - Reduces server load as computations happen on the client-side.
    - Does not require communication with the server for every user interaction.
- **Limitations**:
    - Code is visible to the user and can be modified or misused.

- Heavily dependent on the browser’s capabilities.
- Security concerns if sensitive logic is handled on the client-side.

Server-Side Scripting

- **Definition**: Executes on the web server. The server processes the logic and sends the results (usually HTML) to the client's browser.
- **Languages Used**: PHP, Python, Ruby, Java, Node.js.
- **Key Features**:
    - Handles tasks like database queries, user authentication, and data processing.
    - Keeps sensitive logic and data secure, as the code is executed on the server and not visible to the client.
    - Supports dynamic content generation based on user inputs or database states.
- **Limitations**:
    - Increases server load since every request is processed server-side.
    - Slower for some tasks compared to client-side scripting due to the back-and-forth communication.

Comparison

Aspect	**Client-Side Scripting**	**Server-Side Scripting**
Execution	Runs on the user's browser.	Runs on the web server.
Visibility	Code is visible to the user.	Code is hidden and secure.
Languages	JavaScript, HTML, CSS.	PHP, Python, Ruby, Java, Node.js.
Dependency	Depends on browser compatibility.	Depends on server performance and setup.
Use Cases	Validating forms, animations, and styling.	Database interactions, user authentication.
Speed	Faster for local interactions.	Slower due to server communication.

In conclusion, **client-side scripting** is excellent for enhancing user experiences with interactivity, while **server-side scripting** is vital for secure data handling and dynamic content generation. Both are complementary in modern web development.

CLASS 2

PHP Basics

Objective:

Familiarize students with the fundamental syntax and structure of PHP, including how PHP code is embedded within HTML. By the end of the lecture, students should be able to create a basic PHP script, understand PHP syntax rules, and use PHP tags within HTML.

Definition:

PHP is a scripting language for creating dynamic and interactive web pages. It is embedded in HTML and runs on the server.

- **Explanation**:
  - PHP code is enclosed between <?php ... ?> tags.
  - Statements end with a semicolon ;.
  - Comments can be added using //, #, or /* ... */.
- **Example**:

```
<?php
   // This is a single-line comment
   # This is also a single-line comment
   /* This is a multi-line comment */
```

```
    echo "PHP Basics!";
?>
```

Assignments

1. Write a PHP script to display a message of your choice.
2. Create a script with single-line, multi-line, and hash comments.
3. Write a script to display the sum of two numbers using PHP.
4. Modify the above script to show the result in a formatted sentence, e.g., "The sum of 5 and 10 is 15."
5. Research and list the types of comments in PHP with examples for each.

SOLUTIONS

1. Write a PHP script to display a message of your choice.

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

Explanation:

- The <?php tag is used to start a PHP script.
- The echo statement is used to output text or a message to the browser.
- The message inside the quotes ("Hello, welcome to PHP programming!") will be displayed when the script runs.

2. Create a script with single-line, multi-line, and hash comments.

```php
<?php
   // This is a single-line comment explaining the code below.
    echo "Single-line comment example."; // Another single-line comment.
   # This is a hash comment, which is also a single-line comment.
   echo "Hash comment example.";
 /*
      This is a multi-line comment.
      You can use it to add detailed notes about the code.
   */
   echo "Multi-line comment example.";
?>
```

Explanation:

- //: Denotes a single-line comment.
- #: Also denotes a single-line comment, but it's less commonly used.
- /* ... */: Used for multi-line comments, ideal for longer explanations or temporarily disabling a block of code.

3. Write a script to display the sum of two numbers using PHP.

```
<?php
   $num1 = 5; // First number
   $num2 = 10; // Second number
   $sum = $num1 + $num2; // Calculate the sum
    echo "The sum is: " . $sum; // Display the result
?>
```

Explanation:

- Variables $num1 and $num2 store the two numbers.
- The + operator is used to calculate the sum.
- echo outputs the result with the message "The sum is: ".

4. Modify the above script to show the result in a formatted sentence.

```
<?php
   $num1 = 5; // First number
   $num2 = 10; // Second number
   $sum = $num1 + $num2; // Calculate the sum
    echo "The sum of $num1 and $num2 is $sum."; // Display the
formatted result
?>
```

Explanation:

- Embedded variables ($num1, $num2, $sum) within double quotes are directly parsed and displayed.
- The output will be a clear and formatted sentence, e.g., "The sum of 5 and 10 is 15."

5. Research and list the types of comments in PHP with examples for each.

PHP supports three types of comments:

Single-Line Comments

1. **Using //**

// This is a single-line comment.

echo "Example of single-line comment.";

Any text after // is ignored by the PHP interpreter.

2. **Using #**

# This is another single-line comment.

echo "Example of hash comment.";

Any text after # is also ignored by the PHP interpreter.

Multi-Line Comments

/*

    This is a multi-line comment.

It spans multiple lines and is ideal for larger blocks of notes.

*/

echo "Example of multi-line comment.";

Everything inside /* ... */ is ignored by the interpreter, making it ideal for detailed explanations or temporarily disabling multiple lines of code.

CLASS 3

PHP Variables

- **Objective:**

  Introduce the concept of variables in PHP, including how to declare and assign values to them. Emphasize the importance of variables in storing and manipulating data. By the end, students should understand variable naming rules, types of variables, and how to use variables in PHP scripts.

- **Definition**:
- Variables in PHP are containers used to store data. They are prefixed with a $ symbol.

  Explanation:

  - Variables are dynamic and do not need to be declared with a specific type.

- ○ Variable names are case-sensitive and must start with a letter or an underscore.

Example:

```
<?php
   $name = "Alice";
   $age = 25;
   echo "Name: $name, Age: $age";
?>
```

Assignments

1. Define a variable for your name and display it on the screen.
2. Create variables for your age and city, and display them in a sentence.
3. Write a script to swap values of two variables.
4. Create a variable to hold today’s date and display it.
5. Research and explain why PHP variables start with a $ symbol.

SOLUTIONS

1. Define a variable for your name and display it on the screen.

In PHP, you can define a variable using the $ symbol followed by a name. Here's how you can define a variable for your name and display it:

```
<?php
```

```
// Define a variable for your name

$name = "John Doe";

// Display the variable

echo "My name is: " . $name;

?>
```

Explanation:

- $name is a variable storing the value "John Doe".
- The echo statement outputs the value of the $name variable along with the text "My name is: ".

Output:

```
My name is: John Doe
```

2. Create variables for your age and city, and display them in a sentence.

```
<?php

// Define variables for age and city

$age = 25;

$city = "New York";

 // Display them in a sentence

echo "I am $age years old and I live in $city.";

?>
```

Explanation:

- $age holds the value 25.
- $city holds the string "New York".
- The echo statement uses double quotes, allowing variables to be directly included within the string.

Output:

```
I am 25 years old and I live in New York.
```

3. Write a script to swap values of two variables.

```
<?php

// Define two variables

$a = 10;

$b = 20;

 // Display values before swapping

echo "Before swapping: a = $a, b = $b<br>";

 // Swap the values

$temp = $a;

$a = $b;

$b = $temp;

 // Display values after swapping

echo "After swapping: a = $a, b = $b";
```

```
?>
```

Explanation:

1. $temp is a temporary variable used to store the value of $a.
2. Assign $b to $a.
3. Assign $temp (original value of $a) to $b.

Output:

```
Before swapping: a = 10, b = 20

After swapping: a = 20, b = 10
```

4. Create a variable to hold today’s date and display it.

```
<?php

// Create a variable for today's date

$date = date("Y-m-d");

// Display the date

echo "Today's date is: " . $date;

?>
```

Explanation:

- The date() function in PHP formats the current date and time. The format "Y-m-d" outputs the year, month, and day in the format YYYY-MM-DD.

Output:

Today's date is: 2024-12-03

5. Research and explain why PHP variables start with a $ symbol.

In PHP, all variables are preceded by the $ symbol, and this design choice has several reasons:

1. **Syntax Differentiation:**
    - The $ symbol distinguishes variables from other elements like functions, constants, or keywords in the code. For example, $variable is easily distinguishable from a function call or a string.
2. **Dynamic Typing:**
    - PHP is a dynamically typed language, meaning variables can hold different types of values without declaring their type explicitly. The $ symbol acts as a clear indicator of a variable.
3. **Consistency Across Scope:**
    - Whether a variable is local, global, or a parameter, the $ symbol ensures consistent identification.
4. **Historical Context:**
    - PHP was influenced by the Perl language, which also uses the $ symbol for scalar variables. The design choice in PHP is partly inherited from this legacy.
5. **Readability:**

   - In mixed PHP and HTML code, the $ symbol helps quickly identify PHP variables within the code, improving readability.

6. **Parser Design:**

   - The $ symbol helps PHP's parser identify and process variables efficiently during execution.

CLASS 4

PHP Data Types

- **Objective:**

  Explain the different datatypes available in PHP, such as integers, floats, strings, arrays, and booleans. Students should understand when and why to use each type. By the end, they should be able to identify and utilize appropriate datatypes for storing various types of data in PHP.

- **Definition**:

  PHP data types specify what type of data can be stored and manipulated within a variable. PHP supports several basic types, allowing developers to work with diverse kinds of information.

- **Dynamically Typed**:

PHP determines the type of a variable based on the context in which it's used, making PHP a loosely typed language.

Main Data Types

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

     ```
     $name = "Alice";
     echo "Name: " . $name;
     ```
   - **Common Functions**: strlen($string), strtoupper($string)
2. **Integer**: Non-decimal numbers.
   - **Example**:

     ```
     $age = 25;
     echo "Age: " . $age;
     ```
   - **Properties**: Minimum -2,147,483,648 to maximum 2,147,483,647.
3. **Float (Double)**: Numbers with decimal points.
   - **Example**:

     ```
     $price = 19.99;
     echo "Price: $" . $price;
     ```
4. **Boolean**: Represents true or false values.
   - **Example**:

```
$isStudent = true;
if ($isStudent) {
    echo "Is a student";
}
```

5. **Array**: Holds multiple values in one variable.
    - **Example**:

```
$colors = array("Red", "Green", "Blue");
echo $colors[0];
```

6. **Object**: Instances of classes.
    - **Example**:

```
class Car {
    public $color;
    public function __construct($color) {
        $this->color = $color;
    }
}
$myCar = new Car("Red");
echo $myCar->color;
```

7. **NULL**: Variable with no value.

- **Example**:

```
$var = NULL;

echo $var; // Outputs nothing
```

Assignments:

1. Create a script with variables of each type and print their values.
2. Experiment with type casting.
3. Write a PHP program that calculates and prints the total price of items with different data types.

SOLUTIONS

1. Script with variables of each type and printing their values

```
<?php

// Variables of different types

$integerVar = 25;           // Integer

$floatVar = 3.14;           // Float

$stringVar = "Hello, PHP!";  // String

$booleanVar = true;          // Boolean

$arrayVar = array(1, 2, 3);  // Array

$nullVar = null;             // Null


// Printing their values
```

```php
echo "Integer: $integerVar\n";

echo "Float: $floatVar\n";

echo "String: $stringVar\n";

echo "Boolean: " . ($booleanVar ? 'true' : 'false') . "\n";

echo "Array: " . implode(", ", $arrayVar) . "\n";

echo "Null: " . ($nullVar === null ? 'null' : $nullVar) . "\n";

?>
```

---

2. Experiment with type casting

```php
<?php

// Initial variables

$number = 42;       // Integer

$text = "123.45";   // String representing a number


// Casting to different types

$numberToFloat = (float) $number;  // Integer to Float

$textToInteger = (int) $text;     // String to Integer

$textToFloat = (float) $text;     // String to Float

$numberToString = (string) $number; // Integer to String
```

```
// Printing results

echo "Original integer: $number\n";

echo "Casted to float: $numberToFloat\n";

echo "Original string: $text\n";

echo "Casted to integer: $textToInteger\n";

echo "Casted to float: $textToFloat\n";

echo "Casted integer to string: $numberToString\n";

?>
```

---

3. PHP Program to calculate and print total price of items with different data types

```
<?php

// Item details (price and quantity can have different data types)

$item1Price = 49.99;  // Float

$item1Quantity = 2;   // Integer


$item2Price = "25.50"; // String

$item2Quantity = 3;    // Integer
```

```
$item3Price = 19.75;   // Float
$item3Quantity = "4";  // String

// Calculating total price
$totalPrice =
    ((float)$item1Price * (int)$item1Quantity) +
    ((float)$item2Price * (int)$item2Quantity) +
    ((float)$item3Price * (int)$item3Quantity);

// Printing the total price
echo "The total price of items is: $" . number_format($totalPrice, 2) .
"\n";
?>
```

Output Examples

1. **Script with variables**

```
Integer: 25
Float: 3.14
```

```
String: Hello, PHP!

Boolean: true

Array: 1, 2, 3

Null: null
```

2. **Type casting**

```
Original integer: 42

Casted to float: 42

Original string: 123.45

Casted to integer: 123

Casted to float: 123.45

Casted integer to string: 42
```

3. **Total price program**

```
The total price of items is: $222.49
```

CLASS 5

PHP Operators

- **Objective:**

Introduce PHP operators, including arithmetic, assignment, comparison, and logical operators. Teach students how to use operators to perform calculations and logical operations within PHP scripts. By the end, students should be able to apply operators to manipulate data and make logical decisions in PHP.

- **Definition**:

  Operators are symbols or keywords that specify operations to be performed on operands.

Types of Operators

1. **Arithmetic Operators**: For basic math operations.
   - +, -, *, /, %
   - **Example**:

   ```
   $a = 10;
   $b = 3;
   echo $a + $b;
   ```

2. **Assignment Operators**: Used to assign values.
   - =, +=, -=, *=, /=, %=
   - **Example**:

   ```
   $a = 10;
   $a += 5;
   echo $a; // 15
   ```

3. **Comparison Operators**: Compare values and return a boolean result.
   - ==, !=, >, <, >=, <=
   - **Example**:

```
if ($a == $b) {
    echo "Equal";
}
```

4. **Logical Operators**: Combine multiple conditions.
   - &&, ||, !
   - **Example**:

```
if ($a > 0 && $b < 10) {
    echo "Conditions met";
}
```

Assignment Examples

1. Create a calculator using arithmetic operators.
2. Write a script that compares two numbers.
3. Experiment with logical operators by creating conditions.

SOLUTIONS

1. Create a Calculator Using Arithmetic Operators

```
<?php
```

```
// Calculator functionality

$num1 = 10;

$num2 = 5;


// Arithmetic operations

$sum = $num1 + $num2;

$difference = $num1 - $num2;

$product = $num1 * $num2;

$quotient = $num1 / $num2;

$modulus = $num1 % $num2;


// Display results

echo "Number 1: $num1<br>";

echo "Number 2: $num2<br>";

echo "Sum: $sum<br>";

echo "Difference: $difference<br>";

echo "Product: $product<br>";

echo "Quotient: $quotient<br>";

echo "Modulus: $modulus<br>";
```

```
?>
```

2. Write a Script That Compares Two Numbers

```
<?php
// Input numbers
$num1 = 20;
$num2 = 15;

// Compare numbers
if ($num1 > $num2) {
    echo "$num1 is greater than $num2<br>";
} elseif ($num1 < $num2) {
    echo "$num1 is less than $num2<br>";
} else {
    echo "$num1 is equal to $num2<br>";
}
?>
```

3. Experiment with Logical Operators by Creating Conditions

```
<?php
// Input values
$age = 25;
$hasDrivingLicense = true;

// Logical operator conditions
if ($age >= 18 && $hasDrivingLicense) {
    echo "You are eligible to drive.<br>";
} elseif ($age >= 18 && !$hasDrivingLicense) {
    echo "You need a driving license to drive.<br>";
} else {
    echo "You are not eligible to drive.<br>";
}

// Experiment with OR operator
$isMember = false;
$hasCoupon = true;
```

```
if ($isMember || $hasCoupon) {
    echo "You are eligible for a discount.<br>";
} else {
    echo "You are not eligible for a discount.<br>";
}
?>
```

Each script can be run in a PHP environment like XAMPP or WAMP, and the output will display the results of the operations and conditions.

CLASS 6:

Expressions in PHP

- **Objective:**

  Explain the concept of expressions in PHP, showing how they combine variables, literals, and operators to produce values. Emphasize the role of expressions in assigning values, performing calculations, and making comparisons. By the end, students should be able to create and evaluate expressions in PHP code.

1. Introduction to Expressions in PHP

An expression in PHP is any combination of values, variables, and operators that PHP interprets to produce a value. Expressions are foundational in PHP, as they allow you to perform calculations, assign values, make comparisons, and much more.

Example of an Expression:

$x = 10 + 5;

In this example, 10 + 5 is an expression that evaluates to 15, which is then assigned to the variable $x.

2. Types of Expressions in PHP

PHP expressions can be classified into various types based on their functionality. Here are some commonly used types:

a) Arithmetic Expressions

Arithmetic expressions perform mathematical operations like addition, subtraction, multiplication, division, and modulus.

Example:

$a = 5;

$b = 10;

$sum = $a + $b; // Arithmetic expression that results in 15

b) Assignment Expressions

These expressions assign values to variables using the assignment operator (=) or combined assignment operators (+=, -=, etc.).

Example:

$c = 20;   // Simple assignment

$c += 10; // Equivalent to $c = $c + 10, resulting in 30

c) Comparison Expressions

Comparison expressions are used to compare values. These expressions result in a boolean value: true or false.

Example:

$d = 5;

$e = 10;

$isEqual = ($d == $e); // Comparison expression; evaluates to false

d) Logical Expressions

Logical expressions use logical operators (&&, ||, !) to combine multiple conditions. These are often used in control structures.

Example:

$f = true;

$g = false;

$result = $f && $g; // Logical expression; evaluates to false

e) String Expressions

String expressions use operators to combine or manipulate strings.

Example:

$firstName = "John";

$lastName = "Doe";

$fullName = $firstName . " " . $lastName; // String concatenation; results in "John Doe"

3. Evaluating Expressions in PHP

PHP evaluates expressions from left to right and according to operator precedence and associativity.

Example:

$h = 2 + 3 * 4; // Evaluates to 14, as multiplication has higher precedence than addition

In this case:

- 3 * 4 is evaluated first (resulting in 12)
- Then, 2 + 12 is calculated, resulting in 14.

4. Using Expressions in PHP

Expressions are used in various scenarios in PHP, including:

- **Variable Assignments**

  $i = 5 + 2;

- **Conditional Statements**

  ```
  if ($i > 5) {
      echo "Greater than 5";
  ```

```
}
```

- **Loops**

```
for ($j = 0; $j < 10; $j++) {
    echo $j;
}
```

- **Function Calls**

```
function add($a, $b) {
    return $a + $b;
}


$sum = add(5, 10); // Function call expression
```

5. Operator Precedence and Associativity

When multiple operators are used in an expression, PHP follows specific rules for **operator precedence** (order of operations) and **associativity** (direction in which operations are performed).

- **Operator Precedence**: Defines the order in which different operations are evaluated.
  - Multiplication and division have higher precedence than addition and subtraction.
- **Associativity**: Defines the direction in which an expression is evaluated if operators have the same precedence.

- Most operators in PHP have left associativity (evaluated from left to right).

Example:

$result = 10 + 5 * 2; // Evaluates to 20, because * has higher precedence than +

6. Common Mistakes with Expressions

- **Ignoring Precedence Rules**: Expressions like 10 + 5 * 2 should be written with parentheses for clarity: 10 + (5 * 2).
- **Using Single Equals (=) Instead of Double Equals (==)**: = assigns a value, while == compares values.
- **Not Using Parentheses in Logical Expressions**: Logical expressions like ($a || $b) && $c may yield unexpected results without proper grouping.

7. Summary

- An **expression** is any code that PHP can evaluate to a value.
- Expressions include arithmetic, assignment, comparison, logical, and string types.
- Operator precedence and associativity are crucial for evaluating complex expressions.
- Expressions are used in variable assignments, control statements, loops, and function calls.

8. Practice Examples

Try the following examples to strengthen your understanding of expressions:

1. **Arithmetic Expression Practice**

```
$num1 = 15;
$num2 = 5;
$result = ($num1 + $num2) * 2 - $num2 / $num1;
echo $result;
```

2. **Comparison and Logical Expression Practice**

```
$x = 10;
$y = 20;
$isGreater = ($x < $y) && ($x > 5);
echo $isGreater ? "True" : "False"; // Output should be "True"
```

3. **String Expression Practice**

```
$greeting = "Hello";
$name = "Alice";
echo $greeting . ", " . $name . "!"; // Output should be "Hello, Alice!"
```

9. Homework

Write a PHP script that:

1. Calculates the area of a rectangle using expressions.

2. Checks if a number is positive, negative, or zero using comparison expressions.
3. Combines first and last names into a full name and prints it using string expressions.

SOLUTIONS

1. Calculates the Area of a Rectangle Using Expressions

```
<?php

// Define the length and width of the rectangle

$length = 10;

$width = 5;

// Calculate the area using an arithmetic expression

$area = $length * $width;

// Output the area

echo "The area of the rectangle is: " . $area;

?>
```

Explanation:

- Variables $length and $width store the dimensions of the rectangle.
- The area is calculated using the formula length * width, stored in the $area variable.
- The result is displayed using echo.

2. Checks if a Number is Positive, Negative, or Zero Using Comparison Expressions

```
<?php
// Input number to check
$number = -5;
// Check if the number is positive, negative, or zero
if ($number > 0) {
    echo "The number is positive.";
} elseif ($number < 0) {
    echo "The number is negative.";
} else {
    echo "The number is zero.";
}
?>
```

Explanation:

- A variable $number stores the value to be checked.
- if statements use comparison operators (>, <, ==) to determine the number's state.

- The appropriate message is printed based on the comparison.

---

3. Combines First and Last Names into a Full Name Using String Expressions

```
<?php

// Define first name and last name

$firstName = "John";

$lastName = "Doe";

// Combine names into full name using string concatenation

$fullName = $firstName . " " . $lastName;

// Output the full name

echo "The full name is: " . $fullName;

?>
```

Explanation:

- Variables $firstName and $lastName store the individual's names.
- The . operator concatenates strings, with a space " " in between for readability.
- The combined $fullName is displayed using echo.

CLASS 7

Constants in PHP:

- **Objective:**

  Introduce constants in PHP and explain how they differ from variables. Teach students how to define constants and understand their importance in storing unchangeable values. By the end, students should be able to create constants and use them effectively in their scripts.

- Definition:

  In PHP, a **constant** is an identifier (or a name) for a simple value that cannot be changed during the execution of a script. Constants are useful for defining fixed values that remain the same throughout the program, making your code more readable and easier to maintain.

Examples of common constants: Pi (π = 3.14159), Earth’s gravity (9.8 m/s^2), and mathematical values like EULER (Euler’s number).

Why Use Constants?

- **Fixed Values**: Constants store values that do not change, such as configuration values.
- **Code Readability**: Using descriptive names like TAX_RATE or MAX_USERS is more understandable than using numbers directly in the code.
- **Easy to Update**: Changing the value of a constant in one place will automatically update it everywhere it's used in the code.

Defining Constants in PHP

In PHP, you can define a constant using the define() function or the const keyword.

a) Using define()

The define() function is the traditional way to define constants in PHP.

Syntax:

```
define("CONSTANT_NAME", value);
```

- CONSTANT_NAME: The name of the constant (typically uppercase by convention).

- value: The fixed value assigned to the constant.

Example:

```
define("SITE_NAME", "MyWebsite");

define("TAX_RATE", 0.15);
```

In this example:

- SITE_NAME is a constant holding the value "MyWebsite".
- TAX_RATE is a constant holding the value 0.15.

b) Using const

The const keyword can also be used to declare constants, mainly within classes (more on that in object-oriented programming).

Example:

```
const PI = 3.14159;
```

Rules for Defining Constants

- **Naming**: Constants are usually written in uppercase letters by convention (e.g., PI, MAX_LIMIT).
- **No $ Symbol**: Constants do not use the $ symbol, unlike variables.
- **Global Scope**: Once defined, constants are accessible from anywhere in the script, including inside functions and classes.
- **Immutable**: Constants cannot be changed or undefined once they are declared.

Accessing Constants

Once a constant is defined, you can use it in your code simply by referencing its name.

Example:

```
define("GREETING", "Welcome to PHP!");

echo GREETING; // Output: Welcome to PHP!
```

Constants vs. Variables

Aspect	**Constants**	**Variables**
Changeable	No, value cannot be changed	Yes, value can be reassigned
Scope	Global scope	Local by default
Prefix	No $ symbol	Requires $ symbol
Definition	define() or const	$ symbol with assignment

Predefined Constants in PHP

PHP comes with several built-in constants that you can use directly in your scripts. These constants provide helpful information about the PHP environment.

Examples of Common Predefined Constants:

- PHP_VERSION: The current version of PHP being used.
- PHP_INT_MAX: The largest integer supported.
- E_ALL: All error reporting levels.

Example:

```
echo "PHP version: " . PHP_VERSION;

echo "Maximum integer: " . PHP_INT_MAX;
```

Using Constants in Expressions and Functions

Constants are often used in expressions and function calls to perform calculations or set fixed parameters.

Example:

```
define("DISCOUNT_RATE", 0.1);

$price = 100;

$discountedPrice = $price * (1 - DISCOUNT_RATE);


echo "Discounted Price: " . $discountedPrice; // Outputs 90
```

Constant Arrays (PHP 7+)

Starting from PHP 7, constants can hold arrays, allowing for more flexible data storage.

Example:

```
define("COLORS", ["Red", "Green", "Blue"]);
```

echo COLORS[0]; // Output: Red

10. Case Sensitivity of Constants

By default, constants are case-sensitive. You can make them case-insensitive by passing true as the third parameter in define() (though this is generally discouraged for readability).

Example of Case-Insensitive Constant:

define("LANGUAGE", "PHP", true);

echo language; // Output: PHP

Summary

- Constants hold values that cannot be changed once set.
- Define constants using define() or const.
- Constants do not use the $ symbol and are available globally.
- Use constants for values that do not change and improve code readability.
- PHP provides several predefined constants with valuable system information.

Practice Examples

1. Define a constant for the base URL of a website and print it:

   define("BASE_URL", "https://mywebsite.com");

   echo BASE_URL;

2. Define a constant to store the maximum number of users allowed on a platform and use it in a conditional:

```
define("MAX_USERS", 100);

$currentUsers = 85;

if ($currentUsers < MAX_USERS) {

    echo "You can add more users.";

} else {

    echo "User limit reached.";

}
```

Homework

- Define constants for SALES_TAX, DISCOUNT, and SERVICE_CHARGE and write a PHP program that calculates the total bill for a given purchase amount.
- Experiment with PHP's predefined constants by displaying the PHP_VERSION and PHP_OS constants in a script.

SOLUTIONS

1. Calculate the total bill with constants for SALES_TAX, DISCOUNT, and SERVICE_CHARGE

```
<?php

// Define constants

define('SALES_TAX', 0.07); // 7% sales tax
define('DISCOUNT', 0.10);  // 10% discount

define('SERVICE_CHARGE', 0.05); // 5% service charge


// Given purchase amount

$purchaseAmount = 100.00; // You can change this value to test different amounts


// Calculate discount

$discountAmount = $purchaseAmount * DISCOUNT;


// Calculate subtotal after discount

$subtotal = $purchaseAmount - $discountAmount;


// Calculate sales tax

$salesTaxAmount = $subtotal * SALES_TAX;


// Calculate service charge

$serviceChargeAmount = $subtotal * SERVICE_CHARGE;


// Calculate total bill
```

```
$totalBill = $subtotal + $salesTaxAmount + $serviceChargeAmount;


// Display the results

echo "Purchase Amount: $" . number_format($purchaseAmount, 2) . "<br>";

echo "Discount: -$" . number_format($discountAmount, 2) . "<br>";

echo "Subtotal: $" . number_format($subtotal, 2) . "<br>";

echo "Sales Tax: +$" . number_format($salesTaxAmount, 2) . "<br>";

echo "Service Charge: +$" . number_format($serviceChargeAmount, 2) . "<br>";

echo "Total Bill: $" . number_format($totalBill, 2);

?>
```

Explanation:

We define three constants: SALES_TAX, DISCOUNT, and SERVICE_CHARGE using the define() function. These constants represent the respective rates for sales tax, discount, and service charge.

We set a variable $purchaseAmount to represent the initial amount of the purchase.

We calculate the discount amount by multiplying the purchase amount by the discount rate.

The subtotal is calculated by subtracting the discount from the purchase amount.

We then calculate the sales tax and service charge based on the subtotal.

Finally, we calculate the total bill by adding the subtotal, sales tax, and service charge.

The results are displayed using number_format() to format the amounts to two decimal places.

2. Display PHP's predefined constants for PHP_VERSION and PHP_OS

```
<?php

// Display PHP version

echo "PHP Version: " . PHP_VERSION . "<br>";


// Display PHP OS

echo "PHP OS: " . PHP_OS;

?>
```

Explanation:

We use the predefined constant PHP_VERSION to get the current version of PHP that is running on the server. This constant returns a string representing the version number.

We use the predefined constant PHP_OS to get the operating system on which PHP is running. This constant returns a string representing the OS name.

Both values are displayed using echo, and a line break (<br>) is added for better readability.

CLASS 8

Decision Making and Loops in PHP

- **Objective:**

  Familiarize students with control structures in PHP, focusing on conditional statements (like if, else, and switch) and loops (for, while, do-while). Students should understand how these structures help control program flow. By the end, students should be able to make decisions and iterate over code segments using these constructs.

- Definition

Introduction to Decision Making and Loops

PHP provides structures to control the flow of a program:

- **Decision Making**: Allows PHP to execute different code based on conditions.
- **Loops**: Allow PHP to execute the same block of code multiple times, saving time and reducing redundancy.

1. Decision-Making Statements

Decision-making statements in PHP allow you to perform different actions based on different conditions.

Types of Decision-Making Statements:

1. **if Statement**
2. **if...else Statement**
3. **if...elseif...else Statement**
4. **switch Statement**

1.1 The if Statement

The if statement executes a block of code **only if a specified condition is true**.

Syntax:

```
if (condition) {

    // Code to execute if the condition is true

}
```

Example:

```
$age = 18;

if ($age >= 18) {

    echo "You are eligible to vote.";
}
```

In this example, the message will only display if $age is 18 or more.

1.2 The if...else Statement

The if...else statement adds an **alternative action** if the condition is false.

Syntax:

```
if (condition) {
    // Code to execute if the condition is true
} else {
    // Code to execute if the condition is false
}
```

Example:

```
<?php
$age = 16;
if ($age >= 18) {
    echo "You are eligible to vote.";
} else {
    echo "You are not eligible to vote.";
}

?>
```

1.3 The if...elseif...else Statement

This structure allows multiple conditions to be tested in sequence, and executes code for the **first true condition**.

Syntax:

```
if (condition1) {

   // Code if condition1 is true

} elseif (condition2) {

   // Code if condition2 is true

} else {

   // Code if none of the above conditions are true

}
```

Example:

```
<?php

$score = 85;

if ($score >= 90) {

    echo "Grade: A";
} elseif ($score >= 80) {
    echo "Grade: B";
} elseif ($score >= 70) {
    echo "Grade: C";
} else {
    echo "Grade: F";
}
?>
```

1.4 The switch Statement

The switch statement is a cleaner alternative for checking multiple conditions based on the **value of a single variable**.

Syntax:

```
switch (expression) {
    case value1:
        // Code if expression is equal to value1
        break;
    case value2:
        // Code if expression is equal to value2
        break;
    default:
        // Code if expression does not match any case
}
```

Example:

```
$day = "Tuesday";
switch ($day) {
    case "Monday":
        echo "Start of the week!";
        break;
    case "Tuesday":
        echo "Second day!";
        break;
    default:
        echo "Another day!";
}
```

---

2. Looping Structures in PHP

Loops are used to repeat a block of code as long as a specific condition is true.

Types of Loops:

1. **while Loop**
2. **do...while Loop**
3. **for Loop**
4. **foreach Loop**

2.1 The while Loop

The while loop repeats a block of code **as long as the specified condition is true**.

Syntax:

```
while (condition) {

    // Code to execute

}
```

Example:

```
$count = 1;

while ($count <= 5) {

    echo "Count is: $count<br>";
    $count++;
}
```

In this example, the loop will print the numbers from 1 to 5.

2.2 The do...while Loop

The do...while loop will always execute the code block **at least once**, then repeat as long as the condition is true.

Syntax:

```
do {
    // Code to execute
} while (condition);
```

Example:

```
$count = 1;
do {
    echo "Count is: $count<br>";
    $count++;
} while ($count <= 5);
```

In this example, even if the condition is initially false, the code will run once.

2.3 The for Loop

The for loop is useful when you know in advance how many times you want to execute a statement.

Syntax:

```
for (initialization; condition; increment) {
    // Code to execute
}
```

Example:

```
for ($i = 1; $i <= 5; $i++) {
    echo "Number: $i<br>";
}
```

2.4 The foreach Loop

The foreach loop is used specifically with arrays to iterate over each element.

Syntax:

```
foreach ($array as $value) {
    // Code to execute
}
```

Example:

```
$colors = ["Red", "Green", "Blue"];
foreach ($colors as $color) {
    echo "Color: $color<br>";
}
```

---

3. Using Loops with Conditional Statements

Loops and conditionals are often used together to solve complex problems.

Example: Printing Even Numbers

```
for ($i = 1; $i <= 10; $i++) {
    if ($i % 2 == 0) {
        echo "$i is even<br>";
    }
}
```

---

4. Break and Continue Statements

- **break**: Stops the loop immediately.

- **continue**: Skips the current iteration and moves to the next one.

Example with break and continue:

```
for ($i = 1; $i <= 10; $i++) {
    if ($i == 5) {
        break; // Stops the loop if $i is 5
    }
    if ($i % 2 == 0) {
        continue; // Skips the even numbers
    }
    echo "Number: $i<br>";
}
```

---

5. Practical Examples

1. **Decision Making**

```
$temperature = 30;
if ($temperature > 25) {
    echo "It's a hot day!";
} else {
    echo "The weather is pleasant.";
}
```

2. **Looping**

```
$num = 1;
while ($num <= 3) {
    echo "Count: $num<br>";
    $num++;
}
```

6. Summary

- **Decision-making statements** (if, else, elseif, switch) allow code to execute based on conditions.
- **Looping statements** (while, do...while, for, foreach) repeat code until a specified condition is met.
- **break and continue** statements help control the flow within loops.

7. Homework

- Write a PHP script that checks if a number is positive, negative, or zero using if...elseif...else.
- Write a PHP script that uses a for loop to print the first 10 multiples of 5.
- Write a PHP script that uses a foreach loop to display each element in an array of your favorite movies.

Solutions:

1. Check if a number is positive, negative, or zero

```php
<?php
$number = 10; // You can change this number to test different cases

if ($number > 0) {
    echo "$number is a positive number.";
} elseif ($number < 0) {
    echo "$number is a negative number.";
} else {
    echo "$number is zero.";
}
?>
```

Explanation:

We start by defining a variable $number and assigning it a value.

The if statement checks if the number is greater than zero. If true, it prints that the number is positive.

The elseif statement checks if the number is less than zero. If true, it prints that the number is negative.

The else statement handles the case where the number is neither positive nor negative, which means it must be zero.

2. Print the first 10 multiples of 5 using a for loop

```
<?php

for ($i = 1; $i <= 10; $i++) {

    $multiple = $i * 5;

    echo "5 x $i = $multiple<br>";

}

?>
```

Explanation:

We use a for loop that initializes $i to 1 and continues as long as $i is less than or equal to 10.

Inside the loop, we calculate the multiple of 5 by multiplying $i by 5.

We then print the result in a formatted string, showing the multiplication operation and the result.

The <br> tag is used to create a line break in the output for better readability.

3. Display each element in an array of favorite movies using a foreach loop

```
<?php

$favoriteMovies = array("Inception", "The Matrix", "Interstellar", "The Shawshank Redemption", "Pulp Fiction");

foreach ($favoriteMovies as $movie) {

    echo "$movie<br>";

}

?>
```

Explanation:

We define an array called $favoriteMovies that contains a list of movie titles.

The foreach loop iterates over each element in the $favoriteMovies array.

For each iteration, the current movie is stored in the variable $movie, which we then print.

Again, we use the <br> tag to ensure each movie title appears on a new line.

CLASS 9

Strings in PHP

Objective:

Explain how to work with strings in PHP, covering string creation, accessing characters, searching and replacing text, and formatting. Students will learn about PHP string functions and their applications in text manipulation. By the end, they should be able to perform various operations on strings to manage and format text data in applications.

Definition

1. Introduction to Strings

A **string** is a sequence of characters, such as letters, numbers, and symbols, used to represent text. In PHP, strings are one of the most commonly used data types, as they allow developers to store and manipulate textual data.

Examples of Strings:

$greeting = "Hello, World!";

$quote = 'The quick brown fox jumps over the lazy dog.';

2. Creating Strings in PHP

In PHP, strings can be created using:

1. **Single Quotes (' ')**
2. **Double Quotes (" ")**

Single Quotes

Single-quoted strings are straightforward and do not interpret variables or escape sequences (except for \\ and \').

Syntax:

$singleQuoteString = 'This is a single-quoted string.';

Example:

$name = 'Alice';

$greeting = 'Hello, $name'; // Output: Hello, $name (variable not interpreted)

Double Quotes

Double-quoted strings can interpret variables and special characters, such as \n for a new line or \t for a tab.

Syntax:

$doubleQuoteString = "This is a double-quoted string.";

Example:

$name = "Alice";

$greeting = "Hello, $name"; // Output: Hello, Alice (variable interpreted)

---

3. Accessing Characters in Strings

Each character in a string has an **index** that starts at 0, allowing us to access specific characters using **bracket notation**.

Syntax:

```
$string = "Hello";

echo $string[0];  // Output: H

echo $string[1];  // Output: e
```

Example:

```
$text = "PHP";

echo $text[2];  // Output: P
```

---

4. Common String Functions in PHP

PHP provides a variety of built-in functions to manipulate strings.

4.1 strlen(): Get String Length

The strlen() function returns the length (number of characters) in a string.

Syntax:

```
strlen($string);
```

Example:

```
echo strlen("Hello, World!");  // Output: 13
```

4.2 strpos(): Find Position of a Substring

The strpos() function finds the position of the **first occurrence** of a substring within a string. It returns false if the substring is not found.

Syntax:

```
strpos($string, $substring);
```

Example:

```
$text = "Hello, World!";

echo strpos($text, "World");  // Output: 7
```

4.3 str_replace(): Replace Substring

The str_replace() function replaces all occurrences of a specified substring within a string with a new substring.

Syntax:

```
str_replace($search, $replace, $subject);
```

Example:

```
$text = "Hello, World!";

$newText = str_replace("World", "PHP", $text);

echo $newText;  // Output: Hello, PHP!
```

4.4 strtolower() and strtoupper(): Convert Case

- **strtolower()**: Converts a string to lowercase.
- **strtoupper()**: Converts a string to uppercase.

Example:

```
$text = "Hello, World!";
echo strtolower($text);  // Output: hello, world!
echo strtoupper($text);  // Output: HELLO, WORLD!
```

4.5 ucfirst() and ucwords(): Capitalize First Letter(s)

- **ucfirst()**: Capitalizes the first letter of a string.
- **ucwords()**: Capitalizes the first letter of each word in a string.

Example:

```
$text = "hello world";
echo ucfirst($text);   // Output: Hello world
echo ucwords($text);   // Output: Hello World
```

---

5. Searching for Substrings

To search within strings, PHP offers several useful functions:

- **strpos()**: Returns the position of the first occurrence of a substring.
- **stripos()**: Same as strpos() but is case-insensitive.
- **strstr()**: Finds the first occurrence of a substring and returns the rest of the string from that point.

- **str_contains()**: Checks if a string contains a given substring (PHP 8+).

Example with str_contains():

```
$text = "The quick brown fox";
if (str_contains($text, "fox")) {
    echo "The text contains 'fox'";
}  // Output: The text contains 'fox'
```

---

6. Replacing Parts of Strings

The str_replace() function can replace all occurrences of a substring with another string. This function is particularly useful for text processing and data sanitization.

Example:

```
$sentence = "I like cats.";
$newSentence = str_replace("cats", "dogs", $sentence);
echo $newSentence;  // Output: I like dogs.
```

---

7. Formatting Strings

PHP offers ways to format strings, especially useful for displaying dynamic information:

printf() and sprintf()

- **printf()**: Outputs a formatted string.
- **sprintf()**: Returns a formatted string.

Placeholders:

- %s: String
- %d: Integer
- %f: Floating-point number

Example:

```
$name = "Alice";

$age = 25;

printf("My name is %s, and I am %d years old.", $name, $age);

// Output: My name is Alice, and I am 25 years old.
```

number_format()

The number_format() function formats a number with grouped thousands, commonly used to format currency values.

Example:

```
$amount = 1234.567;

echo number_format($amount, 2);  // Output: 1,234.57
```

8. String Concatenation

In PHP, we use the . operator to concatenate (join) strings together.

Syntax:

```
$string1 . $string2;
```

Example:

```
$firstName = "John";

$lastName = "Doe";

$fullName = $firstName . " " . $lastName;

echo $fullName;  // Output: John Doe
```

---

9. Examples of String Manipulation

Example 1: Greeting Message

```
$name = "Alice";

echo "Hello, " . ucfirst($name) . "!";

// Output: Hello, Alice!
```

Example 2: String Replacement

```
$text = "PHP is fun!";

$newText = str_replace("fun", "awesome", $text);

echo $newText;  // Output: PHP is awesome!
```

Example 3: Formatting Prices

```
$price = 59.99;

echo "Price: $" . number_format($price, 2);

// Output: Price: $59.99
```

---

10. Summary

- **Creating strings**: Use single or double quotes.
- **Accessing characters**: Use bracket notation, e.g., $string[0].
- **Common string functions**: strlen(), strpos(), str_replace(), strtolower(), strtoupper().
- **String concatenation**: Use the . operator.
- **String formatting**: Use printf() and sprintf() for formatted output.

11. Homework

1. Write a PHP script that takes a sentence and replaces a specified word with another word.
2. Write a PHP script that converts a string to uppercase and then counts the number of characters.
3. Write a PHP script that formats a number with thousands separators and two decimal points.

Solutions

1. Replace a specified word in a sentence

```
<?php

$sentence = "The quick brown fox jumps over the lazy dog.";

$wordToReplace = "fox";

$replacementWord = "cat";

// Replace the specified word

$newSentence = str_replace($wordToReplace, $replacementWord,
    $sentence);

echo $newSentence; // Output: The quick brown cat jumps over the
    lazy dog.

?>
```

Explanation:

i. We define a variable $sentence that contains the original sentence.
ii. We specify the word we want to replace ($wordToReplace) and the word we want to use as a replacement ($replacementWord).
iii. The str_replace() function is used to replace occurrences of $wordToReplace with $replacementWord in the $sentence.
iv. The result is stored in $newSentence, which is then printed. The output will show the modified sentence with the specified word replaced.

2. Convert a string to uppercase and count the number of characters

```
<?php

$string = "Hello, World!";

// Convert the string to uppercase

$uppercaseString = strtoupper($string);

// Count the number of characters

$characterCount = strlen($uppercaseString);

echo "Uppercase String: $uppercaseString<br>";

echo "Number of Characters: $characterCount";

?>
```

Explanation:

i. We define a variable $string that contains the original string.
ii. The strtoupper() function is used to convert the entire string to uppercase, and the result is stored in $uppercaseString.
iii. The strlen() function is then used to count the number of characters in the uppercase string, and the result is stored in $characterCount.
iv. Finally, we print both the uppercase string and the character count. The output will show the string in uppercase and the total number of characters.

3. Format a number with thousands separators and two decimal points

```
<?php

$number = 1234567.891;

// Format the number

$formattedNumber = number_format($number, 2, '.', ',');

echo "Formatted Number: $formattedNumber"; // Output:
    Formatted Number: 1,234,567.89

?>
```

Explanation:

i. We define a variable $number that contains the number we want to format.
ii. The number_format() function is used to format the number. The parameters are:
iii. The number to format ($number).
iv. The number of decimal points to display (2 in this case).
v. The character to use as the decimal point (a period .).
vi. The character to use as the thousands separator (a comma ,).
vii. The result is stored in $formattedNumber, which is then printed. The output will show the number formatted with commas as thousands separators and two decimal places.

CLASS 10

Arrays in PHP

Objective:

Introduce arrays in PHP, explaining how to create, access, and manipulate arrays, including multidimensional arrays. Teach students to use arrays for storing multiple values and organizing data. By the end, students should be able to create arrays, access elements, and work with nested arrays for complex data structures.

Definition:

1. Introduction to Arrays

An **array** in PHP is a data structure that allows you to store multiple values within a single variable. Each value in an array is associated with an **index** (or key), which helps in identifying and accessing the values.

- **Syntax:** Arrays in PHP can be created using the array() function or the shorthand [].
- **Types of Arrays:** PHP supports three main types of arrays:
    - **Indexed arrays**: Arrays with numeric indices.
    - **Associative arrays**: Arrays with named keys.
    - **Multidimensional arrays**: Arrays containing other arrays as elements.

---

2. Creating Arrays in PHP

2.1 Indexed Arrays

Indexed arrays use **numeric keys** (starting from 0 by default).

Syntax:

$arrayName = array(value1, value2, value3); // Using array() function

// OR

$arrayName = [value1, value2, value3];   // Using shorthand []

Example:

$fruits = array("Apple", "Banana", "Cherry");

$colors = ["Red", "Green", "Blue"];

In the above example:

- $fruits[0] is "Apple".
- $colors[2] is "Blue".

2.2 Associative Arrays

Associative arrays use **named keys** instead of numeric indices, allowing you to set meaningful keys.

Syntax:

$arrayName = array("key1" => value1, "key2" => value2, "key3" => value3);

// OR

$arrayName = ["key1" => value1, "key2" => value2, "key3" => value3];

Example:

$student = array("name" => "Alice", "age" => 20, "grade" => "A");

$person = ["name" => "John", "email" => "john@example.com"];

In the above example:

- $student["name"] is "Alice".
- $person["email"] is "john@example.com".

2.3 Multidimensional Arrays

A **multidimensional array** is an array that contains other arrays. Each element in the main array can be an array itself, allowing for nested data structures.

Syntax:

```
$multiArray = array(
    array(value1, value2),
    array(value3, value4)
);
```

Example:

```
$students = [
    ["name" => "Alice", "age" => 20],
    ["name" => "Bob", "age" => 22],
    ["name" => "Charlie", "age" => 19]
];
```

In the above example:

- $students[0]["name"] is "Alice".
- $students[1]["age"] is 22.

---

3. Accessing Elements in Arrays

3.1 Accessing Indexed Array Elements

To access elements in an indexed array, use the array variable followed by the **index** inside square brackets [].

Example:

```
$fruits = ["Apple", "Banana", "Cherry"];
echo $fruits[1];  // Output: Banana
```

3.2 Accessing Associative Array Elements

To access elements in an associative array, use the **key** inside square brackets [].

Example:

```
$student = ["name" => "Alice", "age" => 20, "grade" => "A"];
```

```
echo $student["grade"];  // Output: A
```

3.3 Accessing Multidimensional Array Elements

To access elements in a multidimensional array, use multiple sets of square brackets. Each bracket represents a level in the array.

Example:

```
$students = [
    ["name" => "Alice", "age" => 20],
    ["name" => "Bob", "age" => 22]
];
echo $students[1]["name"];  // Output: Bob
```

---

4. Common Array Functions in PHP

PHP offers several built-in functions to work with arrays. Here are some commonly used ones:

4.1 count(): Count Array Elements

The count() function returns the number of elements in an array.

Syntax:

```
count($array);
```

Example:

```
$fruits = ["Apple", "Banana", "Cherry"];

echo count($fruits);  // Output: 3
```

4.2 array_push(): Add Elements to End of Array

The array_push() function adds one or more elements to the end of an array.

Syntax:

```
array_push($array, value1, value2, ...);
```

Example:

```
$colors = ["Red", "Green"];

array_push($colors, "Blue", "Yellow");

print_r($colors);

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

4.3 array_pop(): Remove Last Element

The array_pop() function removes and returns the last element of an array.

Syntax:

```
array_pop($array);
```

Example:

```
$fruits = ["Apple", "Banana", "Cherry"];
```

```
$lastFruit = array_pop($fruits);

echo $lastFruit;  // Output: Cherry
```

4.4 array_merge(): Merge Arrays

The array_merge() function combines two or more arrays into one.

Syntax:

```
array_merge($array1, $array2, ...);
```

Example:

```
$array1 = ["a" => "apple", "b" => "banana"];

$array2 = ["c" => "cherry", "d" => "date"];

$result = array_merge($array1, $array2);

print_r($result);

// Output: Array ( [a] => apple [b] => banana [c] => cherry [d] => date )
```

4.5 in_array(): Check if Element Exists

The in_array() function checks if a specific value exists in an array.

Syntax:

```
in_array($value, $array);
```

Example:

```
$fruits = ["Apple", "Banana", "Cherry"];

if (in_array("Banana", $fruits)) {
    echo "Banana is in the list!";
}  // Output: Banana is in the list!
```

---

5. Multidimensional Arrays: Examples

Example 1: Student Data

```
$students = [
    ["name" => "Alice", "age" => 20, "subjects" => ["Math", "Physics"]],
    ["name" => "Bob", "age" => 22, "subjects" => ["Biology", "Chemistry"]]
];
echo $students[0]["name"];             // Output: Alice
echo $students[1]["subjects"][1];     // Output: Chemistry
```

Example 2: Product List

```
$products = [
    ["id" => 1, "name" => "Laptop", "price" => 1000],
    ["id" => 2, "name" => "Tablet", "price" => 500],
    ["id" => 3, "name" => "Smartphone", "price" => 700]
];
```

```
echo $products[2]["name"];  // Output: Smartphone
```

---

6. Iterating Over Arrays

To loop through arrays, foreach is commonly used.

Indexed Array Example

```
$fruits = ["Apple", "Banana", "Cherry"];

foreach ($fruits as $fruit) {

    echo $fruit . " ";

}

// Output: Apple Banana Cherry
```

Associative Array Example

```
$student = ["name" => "Alice", "age" => 20, "grade" => "A"];

foreach ($student as $key => $value) {

    echo "$key: $value ";

}

// Output: name: Alice age: 20 grade: A
```

Multidimensional Array Example

```
$students = [

    ["name" => "Alice", "age" => 20],
    ["name" => "Bob", "age" => 22]
];
foreach ($students as $student) {
    echo $student["name"] . " is " . $student["age"] . " years old.\n";
}
// Output:
// Alice is 20 years old.
// Bob is 22 years old.
```

---

7. Summary

- **Arrays**: Used to store multiple values in one variable.
- **Types**: Indexed, associative, and multidimensional arrays.
- **Accessing elements**: Use indices for indexed arrays and keys for associative arrays.
- **Common functions**: count(), array_push(), array_pop(), array_merge(), and in_array().
- **Looping through arrays**: foreach loop for accessing elements within arrays.

Understanding arrays and their operations is crucial for handling data effectively in PHP. Arrays allow developers to structure data efficiently and perform complex data manipulations.

HOMEWORK

Assignment 1: Create and Display an Array

Task: Create an array of your favorite fruits and display each fruit on a new line.

Instructions:

Define an array with at least 5 different fruit names.

Use a foreach loop to iterate through the array and print each fruit.

Assignment 2: Array Manipulation

Task: Create an array of numbers and perform the following operations:

Add a new number to the end of the array.

Remove the first number from the array.

Sort the array in ascending order.

Display the modified array.

Instructions:

Define an array with at least 5 numbers.

Use array_push() to add a number.

Use array_shift() to remove the first number.

Use sort() to sort the array.

Assignment 3: Associative Arrays

Task: Create an associative array to store the names and ages of 5 people, then display each person's name and age.

Instructions:

Define an associative array with names as keys and ages as values.

Use a foreach loop to iterate through the array and print each name with its corresponding age.

Assignment 4: Multidimensional Arrays

Task: Create a multidimensional array to store information about students (name, age, and grade) and display the information.

Instructions:

Define a multidimensional array with at least 3 students, each having a name, age, and grade.

Use nested foreach loops to display each student's information.

Assignment 5: Array Functions

Task: Use various PHP array functions to manipulate an array of random numbers.

Instructions:

Create an array of 10 random numbers.

Use array_sum() to calculate the sum of the numbers.

Use count() to find the number of elements in the array.

Use max() and min() to find the maximum and minimum values in the array.

Display the results.

Assignment 6: Filter and Map

Task: Create an array of integers and filter out the even numbers, then square the remaining odd numbers.

Instructions:

Define an array with at least 10 integers.

Use array_filter() to filter out even numbers.

Use array_map() to square the remaining odd numbers.

Display the final array of squared odd numbers.

Assignment 7: Merge and Compare Arrays

Task: Create two arrays and merge them, then find the common elements.

Instructions:

Define two arrays with some overlapping values.

Use array_merge() to combine the arrays.

Use array_intersect() to find the common elements.

Display the merged array and the common elements.

Assignment 8: Array Search

Task: Create an array of colors and search for a specific color.

Instructions:

Define an array with at least 5 color names.

Use in_array() to check if a specific color exists in the array.

Display a message indicating whether the color was found or not.

Assignment 9: Reverse an Array

Task: Create an array of numbers and reverse its order.

Instructions:

Define an array with at least 5 numbers.

Use array_reverse() to reverse the array.

Display the original and reversed arrays.

Assignment 10: Unique Values

Task: Create an array with duplicate values and remove the duplicates.

Instructions:

Define an array with at least 10 elements, some of which are duplicates.

Use array_unique() to remove duplicate values.

Display the original and the array with unique values.

SOLUTIONS

Assignment 1: Create and Display an Array

```
<?php

// Define an array of favorite fruits

$fruits = ["Apple", "Banana", "Cherry", "Date", "Elderberry"];

// Display each fruit on a new line

foreach ($fruits as $fruit) {

    echo $fruit . "<br>";

}

?>
```

Explanation:

i. We define an array called $fruits containing five different fruit names.
ii. We use a foreach loop to iterate through the array and print each fruit, appending a line break (<br>) for formatting.

Assignment 2: Array Manipulation

```
<?php

// Define an array of numbers

$numbers = [5, 3, 8, 1, 4];

// Add a new number to the end of the array
```

```
array_push($numbers, 7);

// Remove the first number from the array

array_shift($numbers);

// Sort the array in ascending order

sort($numbers);

// Display the modified array

echo "Modified Array: " . implode(", ", $numbers);

?>
```

Explanation:

i. We define an array called $numbers with five integers.
ii. We use array_push() to add the number 7 to the end of the array.
iii. We use array_shift() to remove the first element from the array.
iv. We use sort() to sort the array in ascending order.
v. Finally, we display the modified array using implode() to convert the array to a string.

Assignment 3: Associative Arrays

```
<?php

// Define an associative array of names and ages

$people = [

    "Alice" => 25,

    "Bob" => 30,
    "Charlie" => 22,

    "David" => 35,

    "Eve" => 28

];


// Display each person's name and age

foreach ($people as $name => $age) {

    echo "$name is $age years old.<br>";

}

?>
```

Explanation:

i. We define an associative array called $people where names are keys and ages are values.
ii. We use a foreach loop to iterate through the array, printing each name and its corresponding age.

Assignment 4: Multidimensional Arrays

```
<?php

// Define a multidimensional array of students

$students = [

    ["name" => "Alice", "age" => 20, "grade" => "A"],
    ["name" => "Bob", "age" => 22, "grade" => "B"],

    ["name" => "Charlie", "age" => 21, "grade" => "A"]

];

// Display each student's information

foreach ($students as $student) {

    echo "Name: " . $student['name'] . ", Age: " . $student['age'] . ", Grade: " . $student['grade'] . "<br>";

}

?>
```

Explanation:

i. We define a multidimensional array called $students, where each student is represented as an associative array with keys for name, age, and grade.
ii. We use a foreach loop to iterate through the outer array and print each student's information.

Assignment 5: Array Functions

```
<?php

// Create an array of random numbers

$numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

// Calculate the sum of the numbers

$sum = array_sum($numbers);
```

```
// Find the number of elements in the array

$count = count($numbers);

// Find the maximum and minimum values

$max = max($numbers);

$min = min($numbers);

// Display the results

echo "Sum: $sum<br>";

echo "Count: $count<br>";

echo "Max: $max<br>";

echo "Min: $min<br>";

?>
```

Explanation:

i. We define an array called $numbers with ten integers.
ii. We use array_sum() to calculate the sum of the numbers.
iii. We use count() to find the number of elements in the array.
iv. We use max() and min() to find the maximum and minimum values in the array.

Finally, we display the results.

Assignment 6: Filter and Map

```
<?php

// Create an array of integers
```

```
$numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// Filter out even numbers

$oddNumbers = array_filter($numbers, function($num) {

    return $num % 2 !== 0; // Keep odd numbers

});

// Square the remaining odd numbers

$squared ```php

$oddNumbersSquared = array_map(function($num) {

    return $num * $num; // Square the number

}, $oddNumbers);

// Display the final array of squared odd numbers

echo "Squared Odd Numbers: " . implode(", ", $squaredOddNumbers);

?>
```

Explanation:

i. We define an array called $numbers containing integers from 1 to 10.
ii. We use array_filter() to filter out even numbers, keeping only the odd numbers.
iii. We then use array_map() to square each of the remaining odd numbers.

iv. Finally, we display the squared odd numbers using implode() to convert the array to a string.

Assignment 7: Merge and Compare Arrays

```
<?php

// Define two arrays with some overlapping values

$array1 = [1, 2, 3, 4, 5];

$array2 = [4, 5, 6, 7, 8];

// Merge the arrays

$mergedArray = array_merge($array1, $array2);

// Find the common elements

$commonElements = array_intersect($array1, $array2);

// Display the merged array and the common elements

echo "Merged Array: " . implode(", ", $mergedArray) . "<br>";

echo "Common Elements: " . implode(", ", $commonElements);

?>
```

Explanation:

i. We define two arrays, $array1 and $array2, with some overlapping values.
ii. We use array_merge() to combine both arrays into one.
iii. We use array_intersect() to find the common elements between the two arrays.

iv. Finally, we display both the merged array and the common elements.

Assignment 8: Array Search

```
<?php

// Define an array of colors

$colors = ["Red", "Green", "Blue", "Yellow", "Purple"];

// Check if a specific color exists in the array

$searchColor = "Blue";

$found = in_array($searchColor, $colors);

// Display a message indicating whether the color was found or not

if ($found) {

    echo "$searchColor is found in the array.<br>";

} else {

    echo "$searchColor is not found in the array.<br>";

}

?>
```

Explanation:

i. We define an array called $colors with five color names.
ii. We use in_array() to check if a specific color (in this case, "Blue") exists in the array.
iii. We display a message indicating whether the color was found.

Assignment 9: Reverse an Array

```
<?php

// Define an array of numbers

$numbers = [1, 2, 3, 4, 5];

// Reverse the array

$reversedArray = array_reverse($numbers);

// Display the original and reversed arrays

echo "Original Array: " . implode(", ", $numbers) . "<br>";

echo "Reversed Array: " . implode(", ", $reversedArray);

?>
```

Explanation:

i. We define an array called $numbers with five integers.
ii. We use array_reverse() to reverse the order of the array.
iii. Finally, we display both the original and reversed arrays.

Assignment 10: Unique Values

```
<?php

// Define an array with duplicate values

$values = [1, 2, 2, 3, 4, 4, 5, 6, 6, 7];

// Remove the duplicates

$uniqueValues = array_unique($values);
```

```
// Display the original and the array with unique values

echo "Original Array: " . implode(", ", $values) . "<br>";

echo "Array with Unique Values: " . implode(", ", $uniqueValues);

?>
```

Explanation:

i. We define an array called $values that contains some duplicate elements.
ii. We use array_unique() to remove duplicate values from the array.
iii. Finally, we display both the original array and the array with unique values.

These solutions provide a comprehensive overview of how to work with arrays in PHP, covering various operations and functions that are essential for effective array manipulation.

11. PHP Database Integration with MySQL

Objective: Introduce database connectivity in PHP, focusing on how PHP can interact with databases like MySQL to store, retrieve, and manage data. Cover basic database operations (CRUD) using PHP. By the end, students should be able to connect to a database, execute SQL queries, and display data dynamically on a web page.

1. Setting Up MySQL for PHP Integration

Requirements

- **XAMPP or WAMP**: These tools include Apache (web server), MySQL (database), and PHP, allowing local testing of PHP scripts with MySQL databases.
- **phpMyAdmin**: A web-based tool included in XAMPP and WAMP, making it easy to manage MySQL databases.

2. Create Database in MySQL

Create database and table--->For Detail Click Here

2. Connecting PHP to MySQL

PHP offers two main methods to connect to MySQL:

1. **MySQLi (MySQL Improved)**: Supports both procedural and object-oriented programming.
2. **PDO (PHP Data Objects)**: A database-agnostic way to connect to various databases, including MySQL.

Type 1: Connecting Using MySQLi

Example:

The file in which we have to write database connectivity code . It should be named as db/config/db_connect.

Here i take this as db_connect.php

```
<?php
// Database credentials
$servername = "localhost";
$username = "root";
$password = "";
$database = "school";
 // Create connection
$conn = new mysqli($servername, $username, $password,
$database);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

In this example:

- **localhost**: Refers to the server where MySQL is running.
- **root**: Default username for MySQL (use with caution; it's better to create specific users for each application).

- **password**: Left blank by default on local installations like XAMPP, but it should be secure on production servers.
- **student** : It is the database name which you already created in the mysql server before connecting.

Type 2: Connecting Using PDO

PDO is a more flexible choice, supporting multiple databases beyond MySQL. It uses try-catch blocks for error handling.

Example:

```
<?php

try {

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

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

   echo "Connected successfully";

} catch(PDOException $e) {

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

}
```

Comparison of MySQLi and PDO

Feature	**MySQLi**	**PDO**
Database Support	MySQL only	Multiple databases
Named Parameters	No	Yes
Error Handling	Limited	Exception handling
Object-Oriented	Yes	Yes

3. Basic CRUD Operations with PHP and MySQL

CRUD stands for **Create**, **Read**, **Update**, and **Delete** — the four basic operations for manipulating data in a database.

A. Create Operation (INSERT)

The INSERT statement adds new records to a table.

Example:

```
<?php

$sql = "INSERT INTO students (name, age, grade) VALUES ('Alice',
20, 'A')";

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

    echo "New record created successfully";

} else {

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

}
```

This query inserts a new student with name "Alice", age 20, and grade "A" into the students table.

- **Prepared Statements** (for security):

```
<?php

$stmt = $conn->prepare("INSERT INTO students (name, age,
grade) VALUES (?, ?, ?)");

$stmt->bind_param("sis", $name, $age, $grade);

$stmt->execute();

?>
```

B. Read Operation (SELECT)

The SELECT statement retrieves data from one or more tables.

Example:

```
<?php
$sql = "SELECT id, name, age, grade FROM students";
$result = $conn->query($sql);
 if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"]. " - Name: " . $row["name"]. " - Age: " .
$row["age"]. " - Grade: " . $row["grade"]. "<br>";
    }
} else {
    echo "0 results";
}
```

Here, data is retrieved and displayed row-by-row using a loop. This approach allows us to process large datasets efficiently.

C. Update Operation (UPDATE)

The UPDATE statement modifies existing records.

Example:

```
$sql = "UPDATE students SET grade='B' WHERE name='Alice'";
if ($conn->query($sql) === TRUE) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . $conn->error;
}
```

In this example, we change Alice’s grade to "B". Always use prepared statements for user-generated input to avoid SQL injection.

D. Delete Operation (DELETE)

The DELETE statement removes records from the table.

Example:

```
$sql = "DELETE FROM students WHERE name='Alice'";
if ($conn->query($sql) === TRUE) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $conn->error;
}
```

This command removes Alice’s record from the table.

4. Database Security: Preventing SQL Injection

SQL Injection is a security vulnerability that can allow an attacker to manipulate SQL queries by injecting code into form inputs or URL parameters.

Prepared statements are a key defense mechanism against SQL injection.

Example of Using Prepared Statements with MySQLi:

```
$stmt = $conn->prepare("SELECT * FROM students WHERE name
= ?");

$stmt->bind_param("s", $name);

$stmt->execute();

$result = $stmt->get_result();

while($row = $result->fetch_assoc()) {

    echo $row["name"];

}
```

5. Error Handling

- **Error Checking**: Always check the connection and query results.
- **Exception Handling** (PDO): The use of try-catch blocks to catch exceptions can make debugging and error management more effective.

Example:

```
try {
   $stmt = $conn->prepare("INSERT INTO students (name, age) VALUES (:name, :age)");
   $stmt->execute(array(':name' => 'Bob', ':age' => 22));
} catch(PDOException $e) {
   echo "Error: " . $e->getMessage();
}
```

Assignments

1. **Create a Database Connection**: Set up a PHP script to connect to a MySQL database and display a success or error message based on the connection status.
2. **CRUD Operations**:

- o **Insert**: Write a PHP form where users can input their name, age, and grade, which will be added to a students table.
- o **Retrieve**: Develop a PHP script to retrieve and display all records from the students table.
- o **Update**: Create a form that updates a student's grade based on their name.
- o **Delete**: Write a script that deletes a student record based on the student's ID.

3. **Prevent SQL Injection**: Modify your CRUD operations to use prepared statements, demonstrating secure handling of form inputs.
4. **Build a Mini Project**: Create a small web application, like a student management system, where users can add, view, update, and delete student records. Include form validation and basic error handling.
5. **Database Query Assignment**: Write a series of SQL queries to retrieve specific information from the students table, such as students above a certain age, students sorted by grade, and using LIMIT to display the top 3 students.

SOLUTIONS

Assignment 1. Create a Database Connection

This script connects PHP to a MySQL database and displays a success or error message based on the connection status.

```
<?php

// Database credentials

$servername = "localhost";

$username = "root";

$password = "";

$database = "school";



// Create connection

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

// Check connection

if ($conn->connect_error) {

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

}

echo "Connected successfully";

?>
```

Explanation:

- This code establishes a connection to MySQL using MySQLi.
- $servername, $username, $password, and $database are connection details.
- If the connection fails, die() stops execution and displays an error.

Assignment 2. CRUD Operations

A. Insert Operation

This form and PHP script allow users to insert data into the students table.

HTML Form (insert_form.html):

```
<!DOCTYPE html>

<html>

<head>

    <title>Insert Student</title>

</head>

<body>
```

```
   <form action="insert_student.php" method="POST">
      Name: <input type="text" name="name" required><br>
      Age: <input type="number" name="age" required><br>
      Grade: <input type="text" name="grade" required><br>
      <input type="submit" value="Submit">
   </form>
</body>
</html>
```

PHP Script (insert_student.php):

```
<?php
include 'db_connection.php';
$name = $_POST['name'];
$age = $_POST['age'];
$grade = $_POST['grade'];

$sql = "INSERT INTO students (name, age, grade) VALUES ('$name', $age, '$grade')";
```

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

    echo "New record created successfully";

} else {

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

}

$conn->close();

?>
```

- **Explanation**:
    - $_POST captures form input, and INSERT adds a new student.
    - db_connection.php should include the connection script from step 1.
    - $conn->close() closes the connection.

B. Retrieve Operation

This script retrieves and displays all records from the students table.

```
<?php
```

```
include 'db_connection.php';


$sql = "SELECT id, name, age, grade FROM students";

$result = $conn->query($sql);

if ($result->num_rows > 0) {

    while($row = $result->fetch_assoc()) {

        echo "ID: " . $row["id"]. " - Name: " . $row["name"]. " - Age: " . $row["age"]. " - Grade: " . $row["grade"]. "<br>";

    }

} else {

    echo "0 results";

}

$conn->close();

?>
```

- **Explanation**:
    - This code selects all student records, iterates over them with while(), and prints each one.

C. Update Operation

This form and script update a student’s grade based on their name.

HTML Form (update_form.html):

```
<!DOCTYPE html>

<html>

<head>

    <title>Update Student Grade</title>

</head>

<body>

    <form action="update_student.php" method="POST">

        Name: <input type="text" name="name" required><br>

        New Grade: <input type="text" name="new_grade" required><br>

        <input type="submit" value="Update">

    </form>

</body>

</html>
```

PHP Script (update_student.php):

```
<?php

include 'db_connection.php';



$name = $_POST['name'];

$new_grade = $_POST['new_grade'];

$sql = "UPDATE students SET grade='$new_grade' WHERE
name='$name'";

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

    echo "Record updated successfully";

} else {

    echo "Error updating record: " . $conn->error;

}

$conn->close();

?>
```

- **Explanation**:

- This updates the grade where name matches the form input.

D. Delete Operation

This script deletes a student record based on their ID.

HTML Form (delete_form.html):

```
<!DOCTYPE html>

<html>

<head>

   <title>Delete Student</title>

</head>

<body>

   <form action="delete_student.php" method="POST">

      Student ID: <input type="number" name="id" required><br>

      <input type="submit" value="Delete">

   </form>

</body>

</html>
```

PHP Script (delete_student.php):

```
<?php

include 'db_connection.php';

$id = $_POST['id'];

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

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

    echo "Record deleted successfully";

} else {

    echo "Error deleting record: " . $conn->error;

}

$conn->close();

?>
```

- **Explanation**:
    - This code deletes a record where the id matches user input.

---

3. Prevent SQL Injection

To secure CRUD operations, use prepared statements.

Example (Insert with Prepared Statements):

```
<?php
include 'db_connection.php';
$stmt = $conn->prepare("INSERT INTO students (name, age, grade) VALUES (?, ?, ?)");
$stmt->bind_param("sis", $name, $age, $grade);


$name = $_POST['name'];
$age = $_POST['age'];
$grade = $_POST['grade'];
$stmt->execute();
echo "New record created successfully";
$stmt->close();
$conn->close();
?>
```

- **Explanation**:

- bind_param binds data, preventing SQL injection by separating SQL commands from data.

---

4. Mini Project: Student Management System

Combine all CRUD operations above into a single project folder:

1. **Forms** for adding, updating, and deleting.
2. **CRUD scripts** from steps 2A-2D with prepared statements.
3. **Error handling** by checking each operation's success.

Add navigation links or buttons to switch between pages, creating a seamless user interface.

---

5. Database Query Assignment

Here are queries to retrieve specific information from students:

1. **Students above a certain age**:

   ```
   SELECT * FROM students WHERE age > 18;
   ```

2. **Students sorted by grade**:

   ```
   SELECT * FROM students ORDER BY grade DESC;
   ```

3. **Display top 3 students**:

```
SELECT * FROM students ORDER BY grade DESC LIMIT 3;
```

These queries can be run in PHP by using $conn->query() or prepared statements for added security.