PHP

INTRODUCTION:

PHP (PHP: Hypertext Preprocessor) is a server-side scripting language embedded in HTML, used for dynamic content, database management, s development. It supports databases like MySQL, PostgreSQL, Oracle, and SQL Server.

Common Uses of PHP:

- Performs system functions (create, open, read, write, and close files).
- Handles forms (gathers and processes data, sends emails).
- Manages databases (add, delete, modify records).
- Handles cookies (access and set cookie variables).
- Restricts user access to web pages.
- Encrypts data.

Hello World

Displaying "Hello, World!" in PHP:

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

PHP code is processed on the server, and only the HTML output is sent to the client.

COMMENTING YOUR CODE

Comments help in debugging and understanding code. HTML comments:

```
<!-- Insert comment here -->
```

PHP supports multiple comment styles:

```
// Single-line comment
```

```
# Shell-style comment
```

```
/* Multi-line comment */
```

Using comments makes code easier to maintain and understand.

VARIABLES:

Variables store data values.

Example:

```
$city = "New Delhi";
```

Key Points:

- Variables start with $ followed by a meaningful name.
- They cannot start with a number.
- PHP supports variable variables (dynamically named variables).

Example of a variable assignment:

```
$greeting = "Hello";

?>
```

PHP variables can hold different data types like strings, integers, and floating-point numbers.

They are dynamically typed and can store any data type.

Example Assignments:

```
$name = "Suraj John";
$age = 2;
```

Strings are enclosed in quotes, while numbers are not.

Naming Rules:

1. Starts with $, followed by a valid name.
2. Must begin with a letter or underscore.
3. Can contain letters, numbers, and underscores but not special characters like +, -, %, etc.
4. No size limit for variable names.

Case Sensitivity

PHP is case-sensitive, meaning variable names must match exactly.

Example:

```
<?php
$mynameis = "Suraj John";
echo $Mynameis;
?>
```

Incorrect capitalization results in an error:

Warning: Undefined variable: Mynameis in samplecode.php on line 3

PHP Concatenation

Example:

Using your HTML editor, enter the following:

```
<?php
$firstName = "Suraj";
$lastName = "John";
echo "Hello, " . $firstName . " " . $lastName . "!";
?>
```

Output:

Hello, Suraj John!

Destroying PHP Variables

To destroy a variable, pass the variable to PHP's unset() function, as in the following example:

```
<?php
$name = "Smith";
echo $name;
```

```
?>
```

Output:

Smith

In the above example, the variable `$name` holds the value "Smith". The `unset()` function deletes this variable. Since `unset()` is called after `echo`, the firs

Now, if we try to use the variable after `unset()`, we get the following error:

PHP Notice: Undefined variable

CONSTANTS:

A constant is an identifier for a fixed value that cannot be changed during the execution of the script.

Key Features:

- Constants are case-sensitive by default.
- By convention, constant names are written in uppercase.
- A constant name must start with a letter or underscore, followed by letters, numbers, or underscores.
- Once a constant is defined, it cannot be changed or undefined.
- Constants do not use the `$` sign like variables.
- You can retrieve a constant's value using its name directly or with the `constant()` function.

Rules for Defining Constants:

1. Use PHP's `define()` function, which takes two arguments: the constant's name and its value.
2. Constants follow the same naming rules as variables, except they do not use a `$` prefix.

Syntax:

```
<?php
define('SITE_NAME', 'MyWebsite');
?>
```

Example:

```
<?php
define("MAX_USERS", 100);
define("SITE_URL", "https://aksharacs.netlify.app");
echo "Maximum users allowed: " . MAX_USERS . "<br>";
echo "Visit us at: " . constant("SITE_URL"); // Using constant() function
?>
```

Only scalar data types (boolean, integer, float, and string) can be stored in constants.

Differences Between Constants and Variables:

- Constants do not use a `$` prefix, whereas variables do.
- Constants must be defined using `define()`, while variables are assigned with `=`.
- Constants are globally accessible, regardless of scope.
- Constants cannot be redefined or undefined once set.

Valid and Invalid Constant Names:

```
<?php
// Valid constant names
```

```
define("MAX_SCORE", 500);

// Invalid constant names
define("123RANK", "Top Player");  // Starts with a number (Invalid)
define("__CONFIG__", "Settings"); // Reserved naming pattern (Invalid)
?>
```

DATA TYPES:

- PHP is a loosely typed language, meaning variables do not need explicit data type declarations.
- The `gettype()` function returns the data type of a variable.
- The `var_dump()` function displays a variable's value, data type, and size.

Example:

```
<?php
$age = 25;  // Integer
$price = 49.99;  // Float
$name = "Akshara";  // String
$isMember = true;  // Boolean
var_dump($age, $price, $name, $isMember);
?>
```

Output:

int(25) float(49.99) string(5) "Akshara" bool(true)

Types of Data in PHP:

a. **Scalar (Holds a single value):**
   - Integer
   - Float (Double)
   - String
   - Boolean

b. **Compound (User-defined):**
   - Array
   - Object

c. **Special:**
   - Resource
   - NULL

Scalar Data Types

These types store a single value.

Example:

```
<?php
$score = 90;  // Integer
$temperature = 36.6;  // Float
$greeting = "Good Morning";  // String
$isAvailable = false;  // Boolean
echo "Score: " . $score . "<br>";
echo "Temperature: " . $temperature . "<br>";
echo "Greeting: " . $greeting . "<br>";
```

Integer Data Type

An integer is a whole number with no decimal or fractional part. It can be positive, negative, or zero.

The size of an integer depends on the platform, typically 32-bit (max value: 2,147,483,647) or 64-bit (max value: ~9E18).

Example:

```
<?php
$age = 30;  // Integer
var_dump($age);
?>
```

Output:

int(30)

Float (Double) Data Type

A float (or double) is a number that includes a decimal point or is written in exponential form.

Example:

```
<?php
$temperature = 98.6;  // Float
var_dump($temperature);
?>
```

Output:

float(98.6)

Floats are commonly used for precise calculations, such as scientific measurements or financial applications.

String Data Type

A string is a sequence of characters used for storing and manipulating text, including letters, numbers, and special symbols. Strings can be enclosed ir

Declaring Strings

```
<?php
$string1 = "Hello, World!";
$string2 = 'PHP is powerful!';
?>
```

Single vs. Double Quotes

Single-quoted strings are treated literally, while double-quoted strings interpret variables and escape sequences.

```
<?php
$name = "John";
echo 'Hello, $name!'; // Outputs: Hello, $name!
echo "Hello, $name!"; // Outputs: Hello, John!
?>
```

Escape Sequences

Some useful escape sequences in PHP strings:

- \t - Tab
- \" - Double quote
- \\ - Backslash

```
<?php
echo "Hello\nWorld"; // Outputs Hello (new line) World
?>
```

PHP provides a variety of functions to work with strings, including concatenation, length calculation, searching, and manipulation.

1. String Concatenation

String concatenation is the process of joining two or more strings together. In PHP, we use the `.` (dot) operator to concatenate strings.

```
<?php
$name = "John";
$greeting = "Hello, " . $name . "!";
echo $greeting; // Output: Hello, John!
?>
```

Concatenation using `.=` operator:

```
<?php
$message = "Welcome";
$message .= " to PHP!";
echo $message; // Output: Welcome to PHP!
?>
```

2. Finding String Length

PHP provides the `strlen()` function to determine the length of a string.

```
<?php
$text = "Hello, World!";
echo strlen($text); // Output: 13
?>
```

3. Finding Substring Position

To find the position of a substring inside a string, use the `strpos()` function. It returns the index of the first occurrence of the substring or `false` if not

```
<?php
$text = "Hello, World!";
$position = strpos($text, "World");
echo $position; // Output: 7
?>
```

4. Frequently Used String Functions

a) `str_replace()` – Replace Substring in a String

Replaces all occurrences of a search string with a replacement string.

```
<?php
$text = "I love JavaScript!";
$newText = str_replace("JavaScript", "PHP", $text);
```

b) `substr()` – Extract Part of a String

Extracts a portion of a string based on the starting index and optional length.

```
<?php
$text = "Hello, World!";
echo substr($text, 7, 5); // Output: World
?>
```

c) `strtolower()` and `strtoupper()` – Change Case of Strings

Converts a string to lowercase or uppercase.

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

d) `trim()` – Remove Whitespace

Removes leading and trailing whitespace from a string.

```
<?php
$text = "   Hello, World!   ";
echo trim($text); // Output: Hello, World!
?>
```

e) `strrev()` – Reverse a String

Reverses the characters in a string.

```
<?php
$text = "PHP";
echo strrev($text); // Output: PHP
?>
```

f) `explode()` – Convert a String to an Array

Splits a string into an array using a specified delimiter.

```
<?php
$text = "apple,banana,grape";
$fruits = explode(",", $text);
print_r($fruits); // Output: Array ( [0] => apple [1] => banana [2] => grape )
?>
```

g) `implode()` – Convert an Array to a String

Joins array elements into a single string with a specified separator.

```
<?php
$fruits = ["apple", "banana", "grape"];
echo implode(" | ", $fruits); // Output: apple | banana | grape
?>
```

- Use `.` for string concatenation.
- `strlen()` gets string length.
- `strpos()` finds a substring.
- `str_replace()` replaces parts of a string.
- `substr()` extracts substrings.
- `strtolower()` and `strtoupper()` change case.
- `trim()` removes whitespace.
- `strrev()` reverses a string.
- `explode()` splits a string into an array.
- `implode()` joins an array into a string.

These functions make PHP string manipulation easy and efficient!

Boolean Data Type

Definition

A Boolean represents a binary state, either `true` (1) or `false` (0).

Example

```
<?php
$enabled = true;
$disabled = false;
var_dump($enabled, $disabled);
?>
```

Output:

bool(true) bool(false)

Array Data Type

Definition

An array holds multiple values in a single variable. It can store numbers, strings, or objects.

Example

```
<?php
$fruits = array("Apple", "Banana", "Cherry");
var_dump($fruits);
?>
```

Output:

array(3) { [0]=> string("Apple") [1]=> string("Banana") [2]=> string("Cherry") }

Types of Arrays

- **Indexed Array:** Uses numeric indices.
- **Associative Array:** Uses named keys.
- **Multidimensional Array:** Contains nested arrays.

1. Indexed Array

An indexed array is a simple array where elements are stored with a numeric index starting from 0.

```php
<?php
$fruits = array("Apple", "Banana", "Cherry");

// Accessing elements
echo $fruits[0]; // Output: Apple
echo "<br>";

// Looping through an indexed array
foreach ($fruits as $fruit) {
   echo $fruit . "<br>";
}
?>
```

2. Associative Array

An associative array uses named keys instead of numeric indices. It is useful for storing data in key-value pairs.

Example:

```php
<?php
$person = array("name" => "John", "age" => 2, "city" => "Banglore");

// Accessing elements
echo "Name: " . $person["name"] . "<br>";
echo "Age: " . $person["age"] . "<br>";
echo "City: " . $person["city"] . "<br>";

// Looping through an associative array
foreach ($person as $key => $value) {
   echo "$key : $value <br>";
}
?>
```

3. Multidimensional Array

A multidimensional array is an array that contains one or more arrays. It is useful for storing complex data structures.

Example:

```php
<?php
$students = array(
   array("name" => "Akshara", "age" => 5, "grade" => "A"),
   array("name" => "Aron", "age" => 5, "grade" => "B"),
   array("name" => "Suraj", "age" => 2, "grade" => "A")
);

// Accessing elements
echo "Student 1: " . $students[0]["name"] . " - Age: " . $students[0]["age"] . " - Grade: " . $stude
echo "<br>";

// Looping through a multidimensional array
foreach ($students as $student) {
   echo "Name: " . $student["name"] . " - Age: " . $student["age"] . " - Grade: " . $student["grade"
}
?>
```

Summary:

- **Associative Arrays**: Use named keys.
- **Multidimensional Arrays**: Contain multiple arrays.

These array types allow flexible data storage and manipulation in PHP.

Frequently Used Array Functions

- **array_push:** Adds elements to the end of an array.
- **array_pop:** Removes the last element from an array.
- **array_shift:** Removes the first element from an array.
- **array_unshift:** Adds elements to the beginning of an array.
- **array_merge:** Merges two or more arrays.
- **array_slice:** Extracts a portion of an array.
- **array_search:** Searches for a value in an array.
- **array_keys:** Returns all the keys of an array.
- **array_values:** Returns all the values of an array.
- **array_filter:** Filters elements of an array using a callback function.
- **array_map:** Applies a callback function to each element.

Special Data Types

Null Data Type

The special Data type "NULL" represents a variable with no value.

```
<?php
$variable = NULL;

if (is_null($variable)) {
   echo "The variable is NULL";
}
?>
```

Output:

The variable is NULL

Resource Data Type

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.

A common example of using the resource data type is handling files.

```
<?php
$file = fopen("example.txt", "r");

if ($file) {
   echo "File opened successfully";
   fclose($file);
}
?>
```

Output:

File opened successfully

Reading Data from Console

```php
<?php
echo "Enter your name: ";
$handle = fopen("php://stdin", "r");
$name = trim(fgets($handle));
echo "Hello, $name!";
?>
```

Example Console Input & Output:

Enter your name: John

Hello, John!

OPERATORS:

Using different types of operators, values are assigned to variables.

PHP supports more than 50 such operators, ranging from operators for arithmetical operations to operators for logical comparison and bitwise calcula

This section discusses the most commonly used operators.

a. Arithmetic Operators:

+, -, *, /, and % are arithmetic operators.

+ means addition

```php
<?php
$x = 10;
$y = 5;
$z = $x + $y;
echo "Addition: " . $z;
?>
```

- means subtraction

```php
<?php
$z = $x - $y;
echo "Subtraction: " . $z;
?>
```

* means multiplication

```php
<?php
$z = $x * $y;
echo "Multiplication: " . $z;
?>
```

/ means division

```php
<?php
<?php
$z = $x / $y;
echo "Division: " . $z;
?>
```

% means modulus or remainder

```php
<?php
$z = $x % $y;
```

b. Assignment Operators:

The basic assignment operator is (=) sign.

Single equal sign means "assigned to". It does not mean "equal to".

Double equal to sign (==) means "equal to".

Other assignment operators include +=, -=, and .=:

```
<?php
$x = 10;
$x += 5; // Equivalent to $x = $x + 5
echo "x after += : " . $x;
?>
```

```
<?php
$x -= 3; // Equivalent to $x = $x - 3
echo "x after -= : " . $x;
?>
```

```
<?php
$txt = "Hello";
$txt .= " World!"; // Concatenation
echo $txt;
?>
```

c. Comparison Operators:

This operator is used to compare two values.

```
<?php
$x = 4;
$y = 10;
var_dump($x == $y); // false
var_dump($x != $y); // true
var_dump($x > $y);  // false
var_dump($x < $y);  // true
var_dump($x >= $y); // false
var_dump($x <= $y); // true
?>
```

d. Logical Operators:

These operators determine the status of conditions, often used in if-else and loops.

```
<?php
$x = true;
$y = false;
var_dump(!$x); // false
var_dump($x && $y); // false
var_dump($x || $y); // true
?>
```

e. Increment or Decrement Operators:

This operator adds or subtracts from a variable.

```
$x = 5;
echo ++$x; // Pre-increment, outputs 6
echo $x++; // Post-increment, outputs 6 then increases to 7
echo --$x; // Pre-decrement, outputs 6
echo $x--; // Post-decrement, outputs 6 then decreases to 5
?>
```

EXPRESSIONS:

Expressions are fundamental in PHP, as almost everything is an expression. An expression is anything that has a value.

a. Basic Expressions:

Constants, variables, and functions are the simplest expressions. For example:

```
<?php
$a = 5; // 5 is an expression
$b = $a; // $a is also an expression with the value 5
?>
```

Functions return expressions based on their return values:

```
<?php
function foo() {
   return 5;
}
$c = foo(); // Same as $c = 5
?>
```

b. Assignment Expressions:

Assignments also produce a value. For example:

```
<?php
$b = ($a = 5); // Same as $a = 5; $b = 5;
?>
```

c. Increment and Decrement Expressions:

These operators modify values and return expressions based on their type:

```
<?php
$a = 5;
$b = $a++;
$c = ++$a;
?>
```

d. Comparison Expressions:

Expressions evaluating to TRUE or FALSE:

```
<?php
$x = 4;
$y = 10;
var_dump($x == $y); // false
var_dump($x === $y); // false, different types
?>
```

These modify variables using arithmetic operations:

```
<?php
$a = 5;
$a += 3; // Equivalent to $a = $a + 3;
?>
```

f. Ternary Operator:

Conditional shorthand:

```
<?php
$result = ($a > $b) ? "A is greater" : "B is greater";
?>
```

g. Complex Expressions:

Example combining multiple expressions:

```
<?php
function double($i) {
return $i * 2;
}
$b = $a = 5;
$c = $a++;
$d = ++$b;
$f = double($d++);
$g = double(++$e);
$h = $g += 10;
?>
```

PHP - Decision Making

PHP provides various statements to handle decision-making:

- **if...else**: Executes code based on a condition.
- **elseif**: Checks multiple conditions.
- **switch**: Selects one block from multiple choices.

Example: If...Else

```
<?php
$score = 85;
if ($score >= 90) {
   echo "Excellent!";
} elseif ($score >= 75) {
   echo "Good job!";
} else {
   echo "Keep trying!";
}
?>
```

Example: Switch

```
<?php
$day = "Tuesday";
switch ($day) {
```

```
      break;
   case "Friday":
      echo "Weekend is near!";
      break;
   default:
      echo "Another regular day.";
}
?>
```

PHP Loops

Loops help execute code multiple times:

- **for**: Runs a loop for a fixed number of iterations.
- **while**: Runs while a condition is true.
- **do...while**: Runs at least once, then checks the condition.
- **foreach**: Iterates over an array.

Example: For Loop

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

Example: While Loop

```
<?php
$i = 3;
while ($i > 0) {
   echo "Countdown: $i <br>";
   $i--;
}
?>
```

Example: Do...While Loop

```
<?php
$x = 1;
do {
   echo "Iteration: $x <br>";
   $x++;
} while ($x <= 3);
?>
```

Example: Foreach Loop

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

Break exits the loop early, and **Continue** skips an iteration.

Example: Break

```
<?php
for ($i = 1; $i <= 5; $i++) {
   if ($i == 3) break;
   echo "Value: $i <br>";
}
?>
```

Example: Continue

```
<?php
for ($i = 1; $i <= 5; $i++) {
   if ($i == 3) continue;
   echo "Value: $i <br>";
}
?>
```

PHP Functions

Functions in PHP are reusable blocks of code that take input (parameters), process it, and return a result.

PHP provides many built-in functions, such as `strlen()` and `array_push()`, but you can also define your own functions.

Creating a PHP Function

To define a function, use the `function` keyword. Below is an example:

```
<?php
function greetUser() {
   echo "Hello, welcome to PHP functions!";
}
greetUser();
?>
```

PHP Functions with Parameters

Functions can accept parameters to work with different values dynamically.

```
<?php
function greet($name) {
   echo "Hello, $name!";
}
greet("Alice");
?>
```

Passing Arguments by Reference

When passing by reference, the original variable is modified.

```
<?php
function increaseByTen(&$number) {
   $number += 10;
}
$value = 5;
increaseByTen($value);
```

Returning Values from Functions

Functions can return values using the `return` statement.

```
<?php
function multiply($a, $b) {
   return $a * $b;
}
$result = multiply(4, 5);
echo "Multiplication result: $result";
?>
```

Setting Default Values for Function Parameters

You can provide default values for parameters if they are not passed during function calls.

```
<?php
function greetAgain($name = "Guest") {
   echo "Hello, $name!";
}
greetAgain();  // Output: Hello, Guest!
greetAgain("Bob");  // Output: Hello, Bob!
?>
```

Dynamic Function Calls

You can call functions dynamically using variable function names.

```
<?php
function sayHello() {
   echo "Hello, dynamic world!";
}
$func = "sayHello";
$func();
?>
```

File Handling in PHP

PHP provides various functions to handle files, allowing us to create, open, read, write, append, and delete files. Below are the different file operations e

1. Creating a File

To create a file, we use the `fopen()` function with the mode `'w'`. If the file does not exist, it is created automatically.

```
<?php
$file = 'example.txt';
$handle = fopen($file, 'w') or die('Cannot create file');
fclose($handle);
?>
```

2. Opening a File

Files can be opened in different modes such as `'r'` (read), `'w'` (write), and `'a'` (append).

```
<?php
$file = 'example.txt';
$handle = fopen($file, 'r') or die('Cannot open file');
```

3. Reading a File

We use `fread()` to read the file content.

```
<?php
$file = 'example.txt';
$handle = fopen($file, 'r');
$content = fread($handle, filesize($file));
fclose($handle);
echo $content;
?>
```

4. Writing to a File

We use `fwrite()` to write data into a file.

```
<?php
$file = 'example.txt';
$handle = fopen($file, 'w');
fwrite($handle, 'Hello, PHP File Handling!');
fclose($handle);
?>
```

5. Appending to a File

To add content without overwriting, we use mode `'a'` (append).

```
<?php
$file = 'example.txt';
$handle = fopen($file, 'a');
fwrite($handle, "\nAppending new data.");
fclose($handle);
?>
```

6. Closing a File

Always close a file after performing operations using `fclose()`.

```
<?php
fclose($handle);
?>
```

7. Deleting a File

To delete a file, we use the `unlink()` function.

```
<?php
$file = 'example.txt';
unlink($file);
?>
```

Introduction to Form Handling in PHP

Forms are essential in web applications for collecting user input, such as login details, registration data, feedback, and file uploads.

How Form Handling Works?

- Data is submitted to the server via **GET** or **POST**.
- PHP processes the submitted data.
- Data can be stored in a database, file, or used in another process.

Syntax of a Simple Form:

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

Example Form:

Enter Your Name: Submit

Understanding the action Attribute in Forms

The `action` attribute specifies where the form data should be sent after submission.

In this example:

- The form submits data using the `POST` method.
- The form data will be sent to `process.php` for handling.

How process.php Works

When the form is submitted, it sends the data to `process.php`, which processes and displays the result.

Example: process.php

```
<?php
// Check if form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // Get the submitted value
   $username = $_POST['username'];

   // Display the result safely
   echo "<h1>Hello, " . htmlspecialchars($username) . "!</h1>";
}
?>
```

1. Checking the Request Method

The request method determines how form data is sent to the server. PHP checks this using:

```
<?php
   if ($_SERVER["REQUEST_METHOD"] == "POST") {
       // Process the form
   }
?>
```

This ensures that the script runs **only when** the form is submitted using the **POST** method.

2. Retrieving Data from the Form

In PHP, you can get the form data using the `$_POST` superglobal:

```
    $username = $_POST['username'];
?>
```

This retrieves the value entered in the `<input type="text">` field with **name="username"**.

3. Displaying the Data

To display the user's input, use:

```
<?php
    echo "<h1>Hello, " . htmlspecialchars($username) . "!</h1>";
?>
```

Why use `htmlspecialchars()`?

- Prevents **XSS (Cross-Site Scripting) attacks**.
- Escapes special characters like `<`, `>`, and `&`.

Common Uses of the action Attribute

Action Value	Description
`action="process.php"`	Sends data to `process.php` in the same directory.
`action="/submit-form.php"`	Sends data to a PHP script in the root directory.
`action="https://example.com/handle.php"`	Sends data to an external server.
`action=""` (empty)	Submits data to the **same page** that contains the form.

Conclusion

- The `action` attribute determines **where** the form data is sent.
- The `process.php` script handles and processes the form data.
- Using `htmlspecialchars()` prevents **security risks** like XSS.
- Form data can be stored in **databases, files, or used in other processes**.

Types of Form Controls in HTML

1. Input Fields (`<input type="text">`)

Used for entering text.

```
<form method="post">
    <label for="fullname">Full Name:</label>
    <input type="text" id="fullname" name="fullname" placeholder="Enter your name">
    <button type="submit">Submit</button>
</form>
```

Getting Data from Input Fields

The following PHP code retrieves and displays the entered full name:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $fullname = $_POST['fullname'];
    echo "<p><strong>Full Name:</strong> " . htmlspecialchars($fullname) . "</p>";
```

2. Password Fields (`<input type="password">`)

Used for entering passwords securely.

```
<form method="post">
   <label for="password">Password:</label>
   <input type="password" id="password" name="password">
   <button type="submit">Submit</button>
</form>
```

Getting Data from Password Fields

The following PHP code retrieves and displays the entered password:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $password = $_POST['password'];
   echo "<p><strong>Password:</strong> " . htmlspecialchars($password) . "</p>";
}
?>
```

3. Radio Buttons (`<input type="radio">`)

Used for selecting one option from multiple choices.

```
<form method="post">
   <label>
      <input type="radio" name="gender" value="Male"> Male
   </label>
   <label>
      <input type="radio" name="gender" value="Female"> Female
   </label>
   <button type="submit">Submit</button>
</form>
```

Getting Data from Radio Buttons

The following PHP code retrieves and displays the selected gender:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $gender = isset($_POST['gender']) ? $_POST['gender'] : "Not selected";
   echo "<p><strong>Gender:</strong> " . htmlspecialchars($gender) . "</p>";
}
?>
```

4. Checkboxes (`<input type="checkbox">`)

Used for selecting multiple options.

```
<form method="post">
   <label>
      <input type="checkbox" name="hobbies[]" value="Reading"> Reading
   </label>
   <label>
      <input type="checkbox" name="hobbies[]" value="Sports"> Sports
```

```
</form>
```

Getting Data from Checkboxes

The following PHP code retrieves and displays the selected hobbies:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $hobbies = isset($_POST['hobbies']) ? implode(", ", $_POST['hobbies']) : "No hobbies selected";
   echo "<p><strong>Hobbies:</strong> " . htmlspecialchars($hobbies) . "</p>";
}
?>
```

5. Dropdown List (`<select>`)

Used for selecting a single value.

```
<form method="post">
   <label for="country">Country:</label>
   <select id="country" name="country">
      <option value="USA">USA</option>
      <option value="India">India</option>
   </select>
   <button type="submit">Submit</button>
</form>
```

Getting Data from Dropdown List

The following PHP code retrieves and displays the selected country:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $country = $_POST['country'];
   echo "<p><strong>Country:</strong> " . htmlspecialchars($country) . "</p>";
}
?>
```

6. File Upload (`<input type="file">`)

Used to upload files.

```
<form method="post" enctype="multipart/form-data">
   <label for="profile_picture">Upload Profile Picture:</label>
   <input type="file" id="profile_picture" name="profile_picture">
   <button type="submit">Upload</button>
</form>
```

Handling File Uploads

The following PHP code handles file uploads:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST" && !empty($_FILES['profile_picture']['name'])) {
   $file_name = $_FILES['profile_picture']['name'];
   $file_tmp = $_FILES['profile_picture']['tmp_name'];
   move_uploaded_file($file_tmp, "uploads/" . $file_name);
   echo "<p><strong>Uploaded File:</strong> " . htmlspecialchars($file_name) . "</p>";
```

```
}
?>
```

7. Textarea (`<textarea>`)

Used for multi-line text input.

```
<form method="post">
   <label for="comments">Comments:</label>
   <textarea id="comments" name="comments" rows="4" cols="40"></textarea>
   <button type="submit">Submit</button>
</form>
```

Getting Data from Textarea

The following PHP code retrieves and displays the entered comments:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $comments = $_POST['comments'];
   echo "<p><strong>Comments:</strong> " . nl2br(htmlspecialchars($comments)) . "</p>";
}
?>
```

8. Hidden Fields (`<input type="hidden">`)

Used to send hidden values.

```
<form method="post">
   <input type="hidden" name="token" value="12345">
   <button type="submit">Submit</button>
</form>
```

Getting Data from Hidden Fields

The following PHP code retrieves and displays the hidden field value:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $token = $_POST['token'];
   echo "<p><strong>Token:</strong> " . htmlspecialchars($token) . "</p>";
}
?>
```

9. Buttons (`<input type="submit">`, `<input type="reset">`, `<button>`)

Used for submitting and resetting forms.

```
<form method="post">
   <input type="submit" name="submit" value="Submit">
   <input type="reset" value="Reset">
   <button type="submit" name="custom_button" value="clicked">Click Me</button>
</form>
```

Handling Different Buttons in PHP

The following PHP code determines which button was clicked:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   if (isset($_POST['submit'])) {
      echo "<p><strong>Action:</strong> Submit button clicked</p>";
   }
   if (isset($_POST['custom_button'])) {
      echo "<p><strong>Action:</strong> Custom button clicked</p>";
   }
}
?>
```

Complete Registration Form Example

HTML Form (registration.html)

```
<form method="post" action="register.php" enctype="multipart/form-data">
   <h2>Registration Form</h2>

   Full Name: <input type="text" name="fullname"><br><br>
   Password: <input type="password" name="password"><br><br>

   Gender:
   <input type="radio" name="gender" value="Male"> Male
   <input type="radio" name="gender" value="Female"> Female<br><br>

   Hobbies:
   <input type="checkbox" name="hobbies[]" value="Reading"> Reading
   <input type="checkbox" name="hobbies[]" value="Traveling"> Traveling
   <input type="checkbox" name="hobbies[]" value="Music"> Music<br><br>

   Country:
   <select name="country">
      <option value="USA">USA</option>
      <option value="India">India</option>
   </select><br><br>

   Profile Picture: <input type="file" name="profile_picture"><br><br>

   Comments:<br>
   <textarea name="comments" rows="4" cols="30"></textarea><br><br>

   <input type="hidden" name="token" value="abc123">

   <input type="submit" value="Register">
   <input type="reset" value="Reset">
</form>
```

PHP Script (register.php)

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // Getting values
   $fullname = $_POST['fullname'];
   $password = $_POST['password'];
   $gender = isset($_POST['gender']) ? $_POST['gender'] : "Not selected";
   $hobbies = isset($_POST['hobbies']) ? implode(", ", $_POST['hobbies']) : "No hobbies selected";
   $country = $_POST['country'];
   $comments = $_POST['comments'];
```

```
    // Handling File Upload
    if (!empty($_FILES['profile_picture']['name'])) {
        $file_name = $_FILES['profile_picture']['name'];
        move_uploaded_file($_FILES['profile_picture']['tmp_name'], "uploads/" . $file_name);
    } else {
        $file_name = "No file uploaded";
    }

    // Displaying submitted data
    echo "<h2>Registration Details</h2>";
    echo "Full Name: $fullname<br>";
    echo "Password: $password<br>";
    echo "Gender: $gender<br>";
    echo "Hobbies: $hobbies<br>";
    echo "Country: $country<br>";
    echo "Comments: $comments<br>";
    echo "Token: $token<br>";
    echo "Profile Picture: $file_name<br>";
}
?>
```

PHP and MySQL Database Connectivity

PHP and MySQL are widely used together for building dynamic web applications. PHP acts as the backend scripting language, while MySQL is use
Establishing a connection between PHP and MySQL is crucial for performing database operations such as inserting, updating, retrieving, and deleting r

How Database Connectivity Works?

1. **Establish a Connection** → Connect to MySQL using PHP.
2. **Select the Database** → Specify which database to use.
3. **Perform SQL Queries** → Execute commands to store and retrieve data.
4. **Process Data** → Display retrieved information.
5. **Close the Connection** → Free up system resources.

1. Connecting to MySQL and Selecting a Database

PHP provides the `mysqli_connect()` function, which connects to the MySQL server and directly selects a database.

Example: Connecting to MySQL

```
<?php
// Connect to MySQL and select the database
$conn = mysqli_connect("localhost", "root", "password", "test");

// Check if connection was successful
if (!$conn) {
die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully to the database";
?>
```

Explanation:

- `"localhost"` → Server where MySQL is running (use `localhost` if running locally).
- `"root"` → MySQL username.
- `"password"` → MySQL password (replace with your actual password).
- `"test"` → Name of the database to use.

Once connected, you can run SQL queries like `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.

Example: Inserting Data into a Table

```
<?php
$conn = mysqli_connect("localhost", "root", "password", "test");

// SQL query to insert data
$sql = "INSERT INTO users (name, email) VALUES ('John Smith', 'john@gmail.com')";

if (mysqli_query($conn, $sql)) {
echo "New record inserted successfully!";
} else {
echo "Error: " . mysqli_error($conn);
}

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

Explanation:

- We use `mysqli_query($conn, $sql)` to execute the SQL statement.
- `mysqli_error($conn)` displays an error message if the query fails.
- `mysqli_close($conn)` ensures the connection is properly closed.

3. Fetching Data from the Database

Example: Retrieving Data from a Table

```
<?php
$conn = mysqli_connect("localhost", "root", "password", "test");

// SQL query to retrieve records
$sql = "SELECT id, name, email FROM users";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
while ($row = mysqli_fetch_assoc($result)) {
    echo "ID: " . $row["id"] . " - Name: " . $row["name"] . " - Email: " . $row["email"] . "<br>";
}
} else {
echo "No records found!";
}

mysqli_close($conn);
?>
```

4. Updating Records in the Database

Example: Updating a User's Email

```
<?php
$conn = mysqli_connect("localhost", "root", "password", "test");

$sql = "UPDATE users SET email='smith@gmail.com' WHERE name='John Smith'";

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

```
echo "Error updating record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

5. Deleting Records from the Database

Example: Deleting a User

```
<?php
$conn = mysqli_connect("localhost", "root", "password", "test");

$sql = "DELETE FROM users WHERE name='John Smith'";

if (mysqli_query($conn, $sql)) {
echo "Record deleted successfully!";
} else {
echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

6. Closing the Database Connection

Example: Closing Connection

```
<?php
mysqli_close($conn);
?>
```

Conclusion

- **PHP and MySQL enable dynamic web applications** with database connectivity.
- **Use `mysqli_connect()` for modern applications** instead of the deprecated `mysql_connect()`.
- **Always close the connection** using `mysqli_close()` after completing database operations.
- **Use prepared statements** (not covered here) for better security.

This guide provides a foundation for database-driven applications.