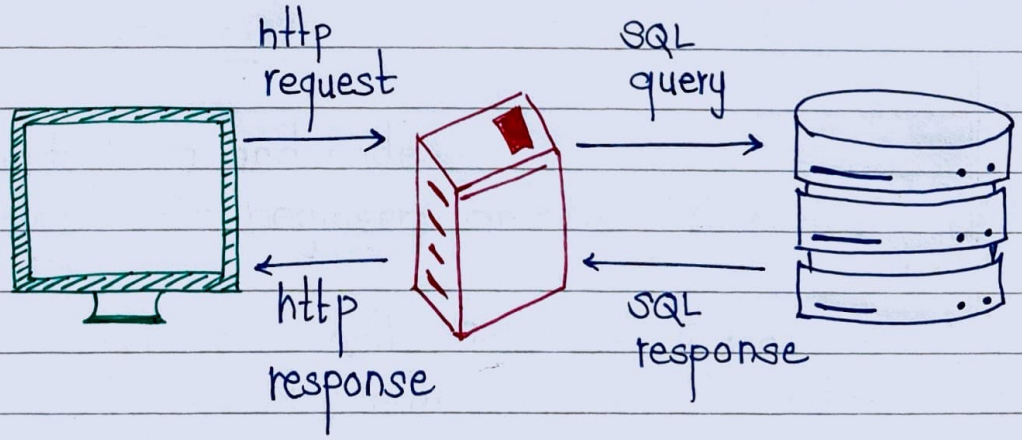
Chapter 1 : Introduction to PHP

1:1 : What is PHP?

PHP is originally stood for Personal Home Page, now stands for PHP : Hypertext Preprocessor - a recursive acronym.
PHP is server-side scripting language, meaning it is executed on the server before the HTML is sent to client's web browser. It allows developers to create dynamic, interactive, and data driven web applications.

• Working of PHP •



Feature of PHP : @curious_programmer

- Open-Source : Free to use and community-supported.
- Cross-platform: Works on Windows, Linux, macOS, Unix.

Loosely typed - Variables don't require explicit data types.

Fast & efficient - Execute code quickly, especially in PHP 7+.

Embedded in HTML : You can fix PHP with HTML seamlessly.

Supports Databases : Built-in support databases like MySQL, PostgreSQL, SQLite.

Extensive Library Support : Thousands of built-in functions and extensions.

1.2 Why Use PHP ?

- Easy to learn and code.
- Excellent for begineers and scalable for professionals.
- Works seamlessly with HTML/CSS/Javascript.
- Large developer community and lots of learning resources.
- Widely used in CMS's like Wordpress, Joomla, Drupal.
- Secure (if used properly with best practices).

Use case examples :

- User authentication (login/logout)
- Contact forms

- Content management Systems
- E-commerce platforms (e.g WooCommerce)
- Real-time data processing (with AJAX + PHP)
- RESTful APIs (with slim, Laravel, etc).

1.3 History and Evolution of PHP.

Understanding PHP's journey helps in appreciating its robustness.

Version	Release	Key Features.
PHP/FI	1995	Created by Rasmus Lerdof for simple web tracking.
PHP3	1998	Added support for database, improved extensibility.
PHP4	2000	Introduced Zend Engine 1.0, better performance.
PHP5	2004	Major revamp with OOP, PDO, better error handling
PHP7	2015	Huge performance boost, scalar type declarations, null coalescing spaceship operator.
PHP8	2020	JIT compiler, union types, match expressions, atributes.

PHP 8.1/8.2 2021-22 Enums, readonly properties, performance improvements.

- Zend Engine:

It is the core engine that parses, compiles, and execute PHP code.

@curious_programmer

1.4 PHP vs others scripting languages

	Feature	PHP	Python	JavaScript
1)	Syntax	C-style	Indentation -based	C-style
2)	Use case	Web apps	General purpose, ML	Full stack web
3)	Server-side	Yes	Yes	Yes
4)	Client-side	No	No	Yes
5)	Learning curve	Easy	Easy	Medium
6)	popular frameworks	Laravel, symfony	Django, Flask	Express.js
7)	Hosting Support	Widely Supported	Moderate	Moderate.

1.5 Installing PHP

Option 1: Using XAMPP (Cross-platform)

- All in one package for Windows, Linux, macOS.
- Includes Apache server, MariaDB (MySQL), PHP, Perl.
- Easy GUI for starting/stopping services.

Download: http://www.apachefriends.org

Option 2: Using WAMP (Windows)

@curious_.programmer

- Designed for Windows users.
- Provided Apache, MYSQL, PHP.

Download: http://www.wampserver.com

Option 3: Using LAMP (Linux)

- Popular on Ubuntu/Linux servers.

bash

```
sudo apt install apache2
sudo apt install php libapache2-mod-php
sudo systemctl restart apache2
```

- After Installation:

- Save your .php files in htdocs (XAMPP) or www (WAMPP)
- Access in browser via http://localhost/filename.php.

1.6 PHP Syntax Basics

PHP code is embedded inside special tags:

```
<?php
    // Your PHP code here
?>
```


- **Rules:**

1. Statement end with semicolon ;.
2. **Comments:**

```
// single-line
/* multi-line */
# shell-style
```

3. **Examples:**

```
<?php
    // This is a comment
    echo "Hello, World!";
?>
```

- **Notes:**

PHP is case-sentive for variable names:
- $Name and $name are different.

Functions and keywords are not case-sensitive.
- echo, Echo, ECHO all work.

1.7 Your First PHP Program

Let's create the classic Hello, World! example.

File: hello.php

```
<!DOCTYPE html>
<html>
<head>
    <title> My First PHP Script </title>
</head>
<body>
<h2> <?php echo "Hello, PHP World!" ; ?> </h2>
</body>
</html>
```

@curious_.programmer

- Output in browser:

Hello, PHP World!

- Steps to Run:

1. Save the file in htdocs/hello.php.
2. Open browser: http://localhost/hello.php.
3. See the result.

1.8 Real-World Uses of PHP :

- 80%+ of websites uses PHP (including Wordpress, Facebook, Wikipedia)
- Used in backend API's, web apps, dashboards, payment gateways, etc.

@curious_.programmer.

Chapter 2 : PHP Basics

2.1 PHP Tags :

PHP files usually saved with a .php extension. The PHP parser looks for code inside PHP tags.

Standard Tags :

@curious_.programmer

```
<?php
    // PHP code goes here
?>
```

These are universally recommended and guaranteed to work on any PHP servers.

Short Open Tags :

```
<?
    echo "short tag";
?>
```

Short tags depend on `short_open_tag` being enabled in php.ini. Many modern servers disable them for security reasons, so avoid them for portable code.

Echo short Tags :

```
<?= "Hi there!" ?>
```

This is a shorthand for:

```
<?php echo "Hi there!"; ?>
```

It is safe and always enable in PHP 5.4+, making it very handy in templates.

- **Best Practices :** @curious_.programmer

Always prefer <?php ?>
Use <?= ?> for simple output in templates.
Avoid <? ?> short tags for portability.

2.2 Comments in PHP :

Comments are essential for writing clean, maintainable code. They do not get executed by the server.

- **Comment styles :**

Style	Syntax	Use
Single line	// or #	Quick notes
Multi line	/*....*/	Block descriptions

- **Why comments matter?**

Explain WHY something is done, not just WHAT is done.

Describe non-obvious logic
Help team members (or your future self)
Makes code readable.

Bad Comment

```
// increment counter
$count++;
```

@curious_.programmer

Good Comment:

```
// Good Count active users visiting during current session
$count++;
```

2.3 Variables in PHP

Variables in PHP are loosely typed, meaning their data type is determined at runtime.

Declaring Variables:

```
$price = 100;
```

You don't have to declare a type like int or string explicitly. PHP figures it out dynamically.

Naming rules:

Must begin with $.
Cannot contain spaces.

Cannot start with a number.
Can use letters, numbers, underscores.
Case sensitive.

Examples of valid variable names:

$counter
$_value
$price123

@curious_.programmer

Examples of invalid names:

$123price
$my price

2.4 Data Types

PHP supports eight basics data types:

Type	Description
String	Textual data, e.g "Hello"
Integer	Whole numbers, e.g. 10
Float	Decimal numbers, e.g. 3.14
Boolean	true/false
Null	Variable with no value
Array	Collection of values
object	Custom class-based structures
Resource	Special handlers (e.g. DB connection)

- Variable inspection

Use var_dump() to see a variable's details:

```
$x = 42;
var_dump ($x);
```

output:

@curious_.programmer

int (42)

This is extremely useful for debugging.

2.5 Constants

Constants provide safety against accidental overwrites:

```
define ("APP_VERSION", "1.0.0");
```

- Rules:

No $prefix
Written in upper case by convention.
Cannot be redefined or unset.

Modern PHP supports:

```
const AUTHOR = "curious_.programmer";
```

✔ When to use constants:

Site name
Tax rates
API keys (through better in environment variables)
Version numbers.

2.6 Operators

Let's explore a few subtle details:

- Arithmetic:

```
$a = 10;
$b = 3;
echo $a % $b;  //1
```


- String operators:

```
$msg = "Hi";
$msg .= "there";
echo $msg;
```

- Array operators:

```
$x = array(1,2,3);
$y = array(4,5,6);
$z = $x + $y;
```

The + operator merges arrays but does not overwrite keys.

2.7 Type Juggling and Type Casting:

Type juggling means PHP will convert automatically:

```
$x = "5" + 10;  // 15 (string converted to int)
```

Type casting means you force it:

```
$price = "9.99";
$priceInt = (int)$price; //9
```

Be careful with automatic conversions!

2.8 Variable Scope

@curious_.programmer

By default, variables defined outside a function are not available inside functions.

```
$name = "PHP";

function showName() {
   global $name;
    echo $name;
}
showName();
```

Static variables:

Static variables remember their value bet-

ween function calls.

```
function counter() {
    static $count = 0;
    $count++;
    echo $count;
}

counter();  //1
counter();  //2
```

@curious_.programmer

2.9 PHP Variable References

In PHP, you can assign variables by reference:

```
$a = 5;
$b = &$a;  //reference
$b = 10;
echo $a;   //10
```

Changing $b also changes $a.

2.10 Predefined Variables (Superglobals)

PHP provides powerful built-in variables called superglobals.

example:

```
echo $_SERVER['HTTP_USER_AGENT'];
```

Variable	Purpose
$_GET	Form data via URL query strings.
$_POST	Form data via POST
$_REQUEST	Combination of GET, POST, COOKIE
$_SERVER	Server and execution env info
$_SESSION	Session variables
$_COOKIE	Cookies
$_FILES	File uploads
$_ENV	Environment variables
$_GLOBALS	All global variables

2.11 Best Practices

@curious_.programmer

- Use <? php ?>.
- Always initialize variables.
- Use descriptive names ($userEmail) instead of ($ue).
- Comment logic, not just syntax.
- Use constants for fixed values.
- keep code blocks short and reusable.
- Validate all user input.
- Use superglobals wisely.

2.12 Practical code examples:

Examples: Type jugling

```
$a = "20" + 5;
echo $a; //25
```

Example: Array union

```
$arr1 = array ("a" => 1, "b" => 2);
$arr2 = array ("b" => 3, "c" => 4);
$result = $arr1 + $arr2;
print_r ($result);
```

@ curious_.programmer

Output:

Array ([a] => 1 [b] => 2 [c] => 4)

Chapter 3 : Control Structures

3.1 Introduction to Control Structures :

Control structures allow you to control the flow of your PHP program. They let you decide :

- When to execute code
- When to skip code
- how many times to repeat code

They are the decision-makers and loops of your program, working together to build powerful logic. In PHP, control structures are quite similar to those in C, Java, Javascript, which helps developers transition smoothly.

3.2 Conditional Statements :

Conditional statements evaluate expressions and determine which block of code to execute. Let's explore each.

3.2.1 if statement :

The simplest control statement :

```
if (condition) {
    // code block runs if condition is true
}
```

Example:

```
$age = 20;
if ($age >= 18) {
    echo "You can vote.";
}
```

3.2.2 if-else statement

Adds an alternative branch:

```
if (condition) {
    // runs if condition true.
} else {
    // runs if condition false.
}
```

Example:


```
$isMember = false;
if ($isMember) {
    echo "Welcome, member!";
} else {
    echo "Please sign up.";
}
```

3.2.3 if-elseif-else ladder

When you need to check multiple conditions:

```
if (condition1) {
    // runs if condition1 true
} elseif (condition2) {
    // runs if condition2 true
} else {
    // fallback
}
```

Example:

```
$score = 75;
if ($score >= 90) {
    echo "Grade A";
} elseif ($score >= 75) {
    echo "Grade B";
} elseif ($score >= 60) {
    echo "Grade C";
} else {
    echo "Grade D";
}
```

@curious_.programmer

3.2.4 switch statement

A cleaner way to handle many discrete cases.

```
switch (variable) {
    case value1:
        // code
        break;
```

```
    case value2:
        // code
        break;
    default:
        // fallback
}
```

Example:

@curious_.programmer

```
$day = "Monday";
switch ($day) {
    case "Monday":
        echo "start of week";
        break;
    case "Friday":
        echo "Weekend is coming!";
        break;
    default:
        echo "Midweek";
}
```

Best Practices with Conditions:

- Always use braces {} even for one-lines for readability.
- Remember break in switch blocks, or code will fall through.
- Avoid deeply nested if-else structures for clarity.

3.3 Looping structures :

Loops are for executing a block of code repeatedly until a condition met. PHP supports four major looping structures.

3.3.1 while loop :

Runs while a condition is true:

```
while( condition){
    // code
}
```

Example : @ curious_programmer

```
$i = 1;
while ($i <=5) {
    echo $i;
    $i++;
}
```

3.3.2 do-while loop :

Similar to while, but guarantees at least one execution:

```
do{
  // code
} while (condition);
```

Example:

```
$i = 1;
do {
  echo $i;
  $i++;
} while ($i <= 5);
```

3.3.3 For loop

Best for count-controlled loops.

```
for (init; condition; update) {
}
```


Example:

```
for ($i = 1; $i <= 5; $i++) {
  echo $i;
}
```

3.3.4 Foreach loop

Made for arrays:

```
foreach ($array as $value) {
    //code
}
```

or

```
foreach ($array as $key => $value){
  //code
}
```

Example:

```
$fruits = ["apple", "banana", "mango"];
foreach ($fruits as $fruit) {
    echo $fruit . "<br>";
}
```

@curious_.programmer

3.4 break and Continue

- **break**

Exits a loop or switch early.

```
for ($i=0; $i<10; $i++){
  if ($i == 5) break;
  echo $i;
}
```

- **Continue**

Skips the current iteration and jumps to the next.

```
for ($i=0; $i<10; $i++) {
  if ($i % 2 == 0) continue;
  echo $i;
}
```

3.5 Nested Control structures :

Control structures can nest inside each other, like:

```
for ($i=1; $i<=3; $i++){
   for ($j=1; $j<=3; $j++){
      echo "($i, $j) ";
   }
}
```

This prints coordinate pairs from (1,1) to (3,3).

Chapter 4 : Functions in PHP

4.1 Introduction to functions:

A function is a block of reusable code that performs a specific task. Functions help make your code.

- **Modular:** break down a large program into smaller pieces.
- **Reusable:** write once, use many times.
- **Readable:** meaningful names clarify purpose.
- **Maintable:** easy to debug and update.

PHP comes with hundreds of built-in functions and you can create your own custom functions too.

4.2 Defining and Calling functions:

Basic Syntax:

```
function functionname() {
    // code to execute
}
```

To call (execute) the function:

```
functionname();
```

Example:

```
function greet() {
    echo "Hello, World!";
}
greet(); // Output: Hello, World!
```

4.3 Functions with Parameters :

Parameters allow you to pass values into a function.

@curious_.programmer

Syntax:

```
function functionName($param1, $param2) {
    // code
}
```

Call it with arguments :

```
functionName(value1, value2);
```

- Default parameters :

If no argument is passed, use a default :

```
function greet($name = "Guest") {
    echo "Hello, $name!";
}
  greet();              // Hello, Guest!
  greet("Curious");     // Hello, Damini!
```

Return Values :

A function can send a result back using return:

```
function multiply ($x, $y) {
    return $x * $y;
}
$result = multiply (4,5); //20
echo $result;
```

@curious_.programmer

Note: Anything after return will not run because return ends the function immediately.

4.4 Variable Scope : Global vs Local

Scope means where a variable can be accessed.

* Local Scope:

Variable declared inside a function are local.
They exist only in that function.

```
function showAge() {
    $age = 25;  //local
    echo $age;
}
showAge();  //25
//echo $age;  //Error! $age not defined outside
```

* Global scope

Variable declared outside functions are global.
They cannot be accessed inside a function directly unless you use global.

```
$city = "Pune";
function printCity() {
    global $city;
    echo $city;
}
printCity();
```

@curious_.programmer

Superglobals:

PHP provides superglobal variables available anywhere

- $_GET, $_POST, $_SERVER, $_SESSION, etc.

These are automatically available to all scopes.

4.5 Built-in Functions:

PHP provides hundreds of built-in functions, broadly grouped as:

- String functions:

strlen($s) → string length
strpos($s, $search) → find substring position

str_replace($search, $replace, $str) → replace text

- **Array Functions:**

array_merge($a1, $a2) → merge arrays.
in_array($val, $arr) → check value.
array_keys → get keys

- **Math Functions:**

abs($x) → absolute value
round($x) → round to nearest
rand($min, $max) → random number

- **Date/Time Functions:**

date($format) → Formatted date
time() → current Unix timestamp

4.8 Variable Functions:

You can use a variable to call a function dynamically.

```
function sayHello (){
    echo "Hello";
}
$func = "sayHello";
$func();    //output: Hello!
```

4.9 Anonymous Functions:

Anonymous functions have no names, and can be stored in variables or passed around.

```
$greet = function($name) {
    return "Hello, $name";
};
echo $greet("curious");  //Hello, curious
```

closures:

Anonymous functions that capture variables from their parent scope.

@curious_.programmer

Examples with closure:

```
$message = "Hi";
$closure = function() use($message) {
    echo $message;
};
$closure();  //Hi
```

This is very common in frameworks (Laravel, etc) for callbacks.

Chapter 5 : Arrays in PHP

5.1 Introduction to Arrays:

An array is a special type of variable that can store multiple values at once under a single name. Instead of creating seprate variables for related items. you can store them together as an array.

Arrays are extremely important in PHP because they:
- allow you to handle lists of data efficiently.
- help organize related data under one umbrella.
- work seamlessly with loops and functions.
- are the foundation for working with forms, JSON data, databases and APIs.

Example use case: @curious_.programmer

Imagine a shopping website wants to store product names.

```
$product1 = "shirt";
$product2 = "Jeans";
$product3 = "shoes";
```

Instead of seprate variables, you could use an array:

```
$products = ["shirt", "Jeans", "shoes"];
```

which is much easier to handle in loops or functions.

5.2 Indexes Arrays

An indexed array uses numeric keys (indices), starting at 0.

- **Declaring Indexed Arrays:**

```
$colors = ["red", "green", "blue"];
```

or using the array() syntax:

```
$colors = array ("red", "green", "blue");
```

- **Accessing Elements:**


```
echo $colors[0]; //red
echo $colors[1]; //green
```

- **Adding/Modifying elements:**

```
$colors[3] = "yellow";
$colors[0] = "pink";
```

- **Iterating Indexed Arrays:**

```
for($i=0; $i< count($colors); $i++) {
    echo $colors[$i] . "<br>";
```

or with foreach:

```
foreach ($colors as $color) {
    echo $color . "<br>";
}
```

5.3 Associative Arrays

An associative array uses named keys instead of numeric indexes, giving the data more meaning.

- **Declaring Associative Arrays:**

```
$person = [
    "name" => "Curious",
    "age" => 22,
    "city" => "Pune"
];
```

@curious_.programmer

- **Accessing Elements:**

```
echo $person["name"];    // curious
```

- **Adding/Modifying Elements:**

```
$person["email"] = "curious@example.com";
$person["city"] = "Mumbai";
```

- **Iterating Associative Arrays:**

```
foreach($person as $key => $value) {
    echo "$key: $value <br>";
}
```

- Why associate arrays ?

- better readability
- meaningful keys
- ideal for representing records (user profiles, products, etc)

5.4 Multidimensional Arrays :

A multidimensional array is an array that contains other arrays, which helps represent table-like data or complex hierarchies.

- 2D Arays (array of arrays)

```
$students = [
    ["Curious", 22, "Pune"],
    ["Priya", 21, "Mumbai"],
    ["Amit", 23, "Delhi"]
];
```

@curious_.programmer

Accessing elements:

```
echo $students[0][0];
echo $students[1][2];
```

- Associative multidimensional arrays

```
$employee = [
  "emp1" => ["name" => "Amit", "salary" => 30000],
  "emp2" => ["name" => "Priya", "salary" => 35000]
];
```

- Accessing elements:

```
echo $employees["emp1"]["salary"];
```

- Herating Multidimensional Arrays:

```
foreach ($students as $student) {
  foreach ($student as $info) {
      echo $info. " ";
  }
  echo "<br>";
}
```

@curious_.programmer

- Why multidimensional arrays?

- store table data
- handle complex data sets
- great for working with forms or database results.

5.5 Array Functions:

PHP provides many built-in array functions to make working with arrays easier. Let's cover some essential ones:

array_merge()

Combines two or more arrays:

```php
$a = [1, 2];
$b = [3, 4];
$c = array_merge($a, $b);
print_r($c);
```

- array_keys()

Get all keys of an array:

```php
$person = ["name" => "curious", "age" => 22];
print_r(array_keys($person));
```

- in_array()

@curious_.programmer

check if a value exists in an array:

```php
$colors = ["red", "green"];
if (in_array("red", $colors)) {
  echo "Found!";
}
```

5.6 Array Iteration:

Iterating arrays means processing each element one by one.

- Using for (best for indexed arrays)

```php
$nums = [10, 20, 30];
for ($i = 0; $i < count($nums); $i++) {
    echo $nums[$i] . "<br>";
}
```

- Iterative associative arrays with keys

```
foreach($person as $key => $value){
    echo "$key : $value <br>";
}
```

- Nested iteration (Multidimension arrays)

```
foreach ($students as $student) {
    foreach ($student as $value) {
        echo $value. " ";
    }
    echo "<br>";
}
```

@curious_programmer

Chapter 6: String handling in PHP

6.1 What is a string?

In PHP, a string is a sequence of characters enclosed within single (') or double (") quotes. A string can obtain:

- letters (A-Z, a-z)
- digits (0-9)
- symbols (e.g @ # $ % ^)
- whitespace
- escape sequences (\n, \t)

Strings are one of the most commonly used data types in PHP, vital for:

- ✓ dynamic web page content
- ✓ form data
- ✓ file names
- ✓ email addresses
- ✓ SQL queries

6.2 Declaring strings

Single-quoted strings

content is treated literally.
Variables are not parsed.
Only \\ and \' are recognized as escape sequences.

```
$str = "This is PHP\'s strength.";
```

- **Double-quoted strings:**

Parses variable inside.
Parses escape sequences.

```
$name = "Curious";
echo "Hello, $name\n";
```

@curious_.programmer

- **String containing quotes:**

```
$str1 = 'She said,"Hello!"';
$str2 = "It's a sunny day.";
```

6.3 Escape Sequences:

Useful within double-quoted strings:

Sequence	Meaning
\n	New line
\t	Tab
\\	Backslash
\"	Double quote
\'	Single quote

6.4 HEREDOC:

Ideal for multi-line strings that parse variables:

```
$name = "Curious";
$text = <<< EOD
Welcome $name,
This is a PHP HEREDOC string.
EOD;

echo $text;
```

6.5 NOWDOC:

Ideal for multi-line strings that do not parse variables (like single quotes):

@curious_.programmer

```
$text = <<< 'EOD'
Welcome $name,
This is a PHP NOWDOC string.
EOD;
echo $text;
```

6.6 String Operators:

- **Concatenation:**

```
$first = "Hello";
$second = "World";
echo $first . " " . $second;
```

- **Concatenation assignment (.=)**

```
$msg = "Hello";
```

```
$msg .= "World";
echo $msg;
```

6.7 Common String Functions:

Let's break down more string functions with examples.

⇒ **strlen()**
Length of a string

```
echo strlen("PHP");
```

@curious_.programmer

⇒ **strpos()**
Position of substring (first occurence)

```
echo strpos("Hello, World", "World");
```

⇒ **substr()**
Extract substring

```
echo substr("Programming", 0, 6);
```

⇒ **str_replace()**
Replace text

```
echo str_replace("bad", "good", "This is bad");
```

⇒ **strtolower() / strtoupper()**
change case

```
echo strtolower("PHP");
echo strtoupper("PHP");
```

⇒ trim():
Remove whitespaces.

```
echo trim("Hello ");
```

⇒ explode():
Convert string → array


```
$data = "apple,banana,mango";
print_r(explode(",", $data));
```

⇒ implode():
convert array → string

```
$fruits = ["apple","banana"];
echo implode("|", $fruits);
```

⇒ strcmp() / strcasecmp()
Strict comparison (case-sensitive / case-insensitive)

```
echo strcmp("abc","abc");
echo strcmp("abc","ABC");

echo strcasecmp("abc","ABC");
```

⇒ substr_count()
Count occurrences

```
echo substr_count("hello hello", "hello");
```

⇒ strrev()

Reverse string

```
echo strrev("PHP");
```

- 6.8 String Formatting

PHP provides printf() and sprintf() for formatted strings.

```
$price = 10.5;
printf("Price: %.2f", $price);
```


sprintf() returns string instead of printing:

```
$msg = sprintf("Price: %.2f", $price);
```

- 6.9 String and Security

Always sanitize strings from user input to prevent:

- HTML injection → use htmlspecialchars()
- SQL injection → use prepared statements.

```
$userInput = '<script>alert(1)</script>';
echo htmlspecialchars($userInput);
```

Chapter 7: Forms and User Input in PHP

Introduction:

PHP's most powerful feature is its ability to build dynamic websites that accept user input. This is usually done through HTML forms, which allow you to collect data, process it on the server, and provide personalized responses.

Without user input, web pages would be static, boring. Forms make them interactive — the basis for logins, shopping carts, feedback, payments, and more.

7.2 HTML form refresher:

@curious_.programmer

A basic HTML form looks like this:

```
<form action="process.php" method="post">
    <label> Username: </label>
    <input type="text" name="username">
    <input type="submit" value="send">
</form>
```

- **key elements of a form:**

action :- where to send the data

method :- how to send the data (GET or POST)

input fields: Where the user types

submit button : triggers the send.

7.3 GET vs POST (In Depth)

Feature	GET	POST
Data location	Visible in URL	hidden in body
Data length	limited	unlimited
Bookmarks	Can bookmark	Cannot bookmark
Security	less secure	more secure
Use case	search, filters	login, password, forms with sensitive data.

- Example using GET: @curious_.programmer

```
<form method="get" action="search.php">
    <input type="text" name="query">
    <input type="submit">
</form>
```

When you search for query=php, the browser shows :

search.php?query=php

Example using POST :

```
<form method="POST" action="register.php">
    <input type="text" name="username">
    <input type="password" name="password">
    <input type="submit">
</form>
```

POST data will not appear in the URL.

7.4 PHP Superglobals @curious_.programmer

PHP automatically gives you superglobal variables for working with user data:

$_GET
$_POST
$_REQUEST
$_FILES (for uploads)
$_COOKIE
$_SESSION

Superglobals are always available in any scope, without global keywords.

7.5 Handling Form Data

When you submit the form, you access the data with these superglobals.

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

If a field is missing, it will give a warning, so best practice is to check:

```
if(isset($_POST['username'])) {
    $username = $_POST['username'];
}
```

@curious_.programmer

7.6 Validating User Input:

Validation means checking that the data meets certain criteria before using it.

Type	Example
Required	Cannot be empty
Type check	numbers only, email only
Length check	passwords at least 8 characters.
Range check	age between 18-99
pattern check	match a regular expression

- **Example: Validate email**

```
if(!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Invalid email";
}
```

Example: Validate numeric

```
if(! is_numeric($_POST["age"])) {
    echo "Age must be a number";
}
```

7.7 Sanitizing User input: @curious_.programmer

Sanitization means cleaning up to data to remove harmful or unexpected characters.

Function	Purpose
htmlspecialchars()	prevents HTML or JS code from running.
strip_tags()	removes all HTML tags.
trim()	removes spaces around text.
filter_var() with FILTER_SANITIZE_STRING	removes invalid characters.

Example:

```
$name = trim($_POST["name"]);
$name = strip_tags($name);
$name = htmlspecialchars($name);
```

7.8 Example : User registration

- HTML form:

```
<form method = "post">
    <input type = "text" name="email">
    <input type = "password" name = "pass">
    <input type = "submit">
</form>
```

- PHP :


```
if ($_SERVER["REQUEST_METHOD"] == "POST"){
  $email = trim($_POST["email"]);
  $pass = $_POST["pass"];

   if (! filter_var($email, FILTER_VALIDATE_EMAIL)){
      echo "Invalid email";
   } elseif (strlen($pass) < 8){
      echo " password must be 8+ chars";
   } else {
      echo "Registration Successful!";
   }
}
```

7.9 HTML injection & XSS

XSS (cross-site scripting) happens when an attacker puts harmful HTML/JS in your form fields.

```
echo "<h1>Welcome, " . $_POST["username"] . "</h1>";
```

if Someone types

```
<script> alert ('hacked') </script>
```

-it will execute!

@curious_.programmer

Prevent it:

```
echo "<h1> Welcome, ".htmlspecialchars ($_POST ["username"])
. "</h1>";
```

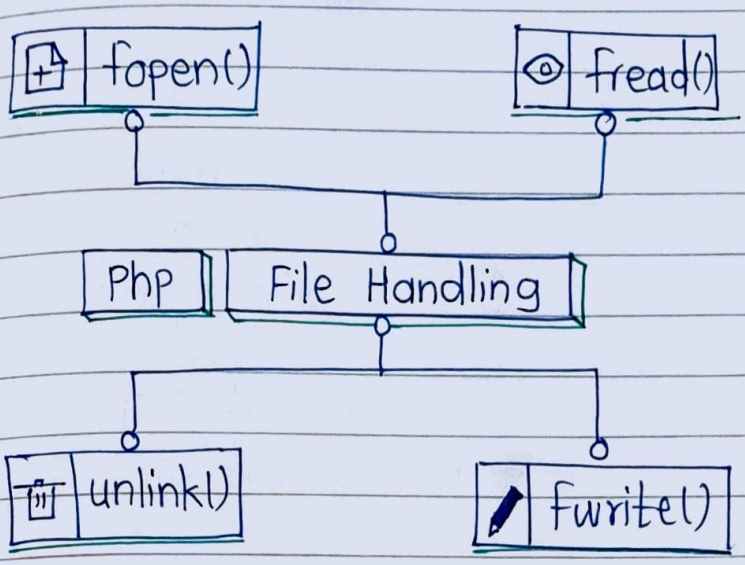
7.10 Multiple Inputs/Arrays

For checkboxes or multi-select:

```
<form method = "post">
    <input type= "checkbox" name= "hobbies[]" value=
    "music"> Music
    <input type="checkbox" name= "hobbies[]" value= "music"
    Sports"> Sports
    <input type ="checkbox" name= " hobbies[]" value=
    " travel" Travel
    <input type= "submit">
</form>

if(! empty ($_POST["hobbies"])) {
    foreach ($_POST["hobbies"] as $hobby) {
        echo $hobby. "<br>";
    }
}
```

Chapter 8 : File Handling in PHP



8.1 Introduction

A huge strength of PHP is its ability to read, write, create, and manipulate files on the server.

This makes PHP extremely useful for:

- ✓ Saving user data
- ✓ logs
- ✓ uploading and storing images
- ✓ generating reports
- ✓ Creating Configuration files

File handling is a core web development skill, as virtually every web app iteracts with files in some form.

8.2 Basic File Operations

PHP supports standard file operations:

- create a file
- read from a file
- write to a file
- append to a file
- delete a file
- check if a file exists
- change file permissions.

These are done using file system functions.

8.3 Opening and closing Files

The primary function to open a file is fopen():

* `$handle = fopen("example.txt","r");`

mode	meaning
r	read-only
w	write, overwrite
a	append
x	create new, error if exists
r+	read / write
w+	read/write, overwrite
a+	read/write, append

always close the file:

```
fclose($handle);
```

8.4 Reading Files

There are multiple ways to read:

⇒ fgets() → reads one line at a time

```
$handle = fopen("data.txt", "r");
while ($line = fgets($handle)) {
    echo $line. "<br>";
}
fclose($handle);
```

@curious_.programmer

⇒ fread() → reads a specific number of bytes:

```
$handle = fopen("data.txt", "r");
$content = fread($handle, filesize("data.txt"));
echo $content;
fclose($handle);
```

⇒ file() → reads entire file into an array of lines:

```
$lines = file("data.txt");
foreach ($lines as $line) {
```

```
    echo $line . "<br>";
}
```

⇒ file_get_contents() → Quickly reads the entire file as a string.

@curious_programmer

```
$data = file_get_contents("data.txt");
echo $data;
```

8.4 Writing to Files

You can write using fwrite() or file_put_contents().

Example with fwrite():

```
$handle = fopen("data.txt", "w");
fwrite($handle, "Hello World!");
fclose($handle);
```

Example with file_put_contents():

```
file_put_contents("data.txt", "PHP is awesome!");
```

This is a simpler one-line method to write files.

8.5 Appending to files

Instead of overwriting, you can append with the mode 'a'.

```
$handle = fopen("log.txt", "a");
fwrite($handle, "New log entry\n");
fclose($handle);
```

8.6 Deleting Files

Use unlink() to delete a file:

```
if (file_exists("old.txt")){
    unlink("old.txt");
    echo "File deleted";
}else{
    echo "File not found";
}
```


8.7 File Uploads via forms

PHP makes uploading files easy.

Example form :

```
<form method="post" enctype="multipart/form-data">
    <input type="file" name="myfile">
    <input type="submit" value="Upload">
</form>
```

Handling upload in PHP :

```
if($_FILES["myfile"]["error"] == 0) {
    move_uploaded_file(
```

```
        $_FILES["myfile"]["tmp_name"],
        "uploads/" . $_FILES["myFile"]["name"]
    );
    echo "Upload successfully";
}
```

8.8 File Permissions

On Linux systems, you might need to change file permissions with chmod.

PHP can check permissions with: @curious_.programmer

```
if (is_writable("data.txt")) {
    echo "Can write to file";
}
```

or

```
if(is_readable("data.txt")){
    echo "Can read file";
}
```

8.9 Checking File Existence

Always verify a file exists before reading or deleting:

```
if( file_exists ("data.txt")) {
    echo "File exists!";
}
```

8.10 Working with Directories

PHP also supports directory functions:

- mkdir() to create a directory.
- rmdir() to remove a directory.
- opendir() / readdir() to list contents.

Example: @curious_.programmer

```
mkdir("photos");
```

8.11 Common File Handling Functions:

Functions	Description
fopen()	open files
fclose()	close file
fread()	read bytes
fwrite()	write
file_get_contents()	quick read
File_put_contents()	quick write
unlink()	delete

filesize()	get file size
rename()	rename file

@curious_.programmer

Chapter 9 : Sessions and Cookies

9.1 Introduction

Web applications often need to remember users and persist data across multiple pages – like login credentials, preferences, or shopping cart contents since HTTP is a stateless protocol, it does not maintain user information by default.

@curious_programmer

To overcome this, PHP provides :

- Sessions – Store data on the server
- Cookies – Store data on the client's browser

These mechanisms are essential to build interactive, personalized, and secure web applications.

9.2 What is a session ?

A session allows data to be stored on the server and linked to a specific user via a unique session ID. This ID is usually sent to the user's browser in a cookie called PHPSESSID.

Key Features :

- Server-side storage (more secure)

- Temporary by default (lasts untill browser closes or manually destroyed)
- Good for sensitive data (like login credentials)

Example: Starting a session

```
<?php
    session_start();

    $_SESSION["username"] = "ABC";
    $_SESSION["role"] = "admin";
?>
```


Accessing Session Data:

```
<?php
session_start();
echo "Welcome, ". $_SESSION["username"];
?>
```

Destroying a session:

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

9.3 What is a cookie?

A cookie is a small piece of data stored on the client side (user's browser). It is automatically sent with every request to the server, allowing PHP to access it.

Key Features:

- Stored on user's browser.
- Can persist for a specific duration (days, months).
- Best for non-sensitive data (preferences, theme, etc).

- **Setting a cookie:**

@curious_.programmer

```
<?php
setcookie("user", "ABC", time() + (86400 * 7));
?>
```

must be sent before any HTML output.

- **Reading a cookie:**

```
<?php
echo $_COOKIE["user"];
?>
```

- **Deleting a cookie:**

```
<?php
```

```
setcookie("user", "", time() - 3600);
?>
```

9.4 Difference between sessions and cookies:

Feature	Sessions	Cookies
storage	Server	client (Browser)
security	More secure	Less secure
Size Limit	Large (as per server)	~ 4kB
expiry	Ends with browser or manual	Can be long-term
Visibility	Hidden from user	Visible in browser

9.5 Use cases

@curious_.programmer

Task	Recommended.
Login and Authentication	Session
"Remember Me" Functionality	Cookie
Language or Theme selection	Cookie

Cart/checkout Flow in E-commerce

Session

9.6 Real-World Example - Login System

HTML Form:

```
<form method = "post">
 Email: <input type= "text" name = "email">
 Password: <input type = "password" name = "password">
 <input type="submit" name= "login" value= "Login">
</form>
```

@curious_programmer

PHP Logic:

```
<?php
 session_start();
 $email = $_POST['email'];
 $password = $_POST['password'];

 if($email === "admin@example.com" && $password
    === "1234") {
    $_SESSION['user'] = $email;
    echo "Login successfully!";
 } else {
    echo "Invalid credentials";
 }
?>
```

9.7 Real-World Example: "Remember Me" with cookies

Login Form with Remember Option:

```
<form method="post">
    Email: <input type="text" name="email">
    <input type="checkbox" name="remember"> Remember Me
    <input type="submit" value="Login">
</form>
```

PHP Logic:

```
<?php
if (isset ($_POST['remember'])){
  setcookie("email", $_POST['email'], time() + (86400 *
  30)); //
}
?>
```

@curious_.programmer

To pre-fill form on next visit:

```
<input type="text" name="email" value="<?php echo
  $_COOKIE['email'] ?? ''; ?>">
```