UNIT -3
REUSING CODE AND WRITING FUNCTIONS

Advantages of Reusing Code

Code reusability is the practice of writing code in such a way that it can be reused in different parts of a program without rewriting it. This enhances the efficiency, maintainability, and readability of software projects.

1. Reduction of Code Duplication

- Reusing code helps eliminate redundancy. Instead of writing the same code multiple times, developers can define a reusable function or file and call it whenever needed.
- This leads to cleaner and shorter codebases.

2. Improved Maintainability

- When the code is reused from a single source, any changes or bug fixes made to that part of the code are automatically reflected in all the places where it is used.
- This simplifies the process of maintenance and reduces the chance of errors.

3. Enhanced Readability and Organization

- Reused code is often written in the form of functions, classes, or modules.
- This modular approach helps in organizing the code logically, making it easier to understand, debug, and extend.

4. Faster Development

- Code reuse leads to quicker development as it saves time by avoiding repetitive coding.
- Developers can focus on implementing new features instead of rewriting already available functionalities.

5. Improved Testing and Debugging

- When a particular piece of code is reused, it can be tested independently.
- Once it is verified to work correctly, it can be confidently reused in multiple areas without additional testing, thereby improving the reliability of the application.

6. Encourages Modular Programming

- Reusability is closely associated with modular programming. In PHP, separate modules (files or functions) can be created for specific tasks and reused across the application.
- This results in better program structure and modularity.

include and require in PHP

- In PHP, modular programming is encouraged through the use of external files.
- The include and require statements are used to insert the content of one PHP file into another before the server executes it.
- These functions help in **code reuse**, **modularity**, and **maintenance** of PHP scripts.

Syntax

```
include 'filename.php';

require 'filename.php';
```

Feature	include	require
Error Handling	Generates a **warning** if file not found	Generates a **fatal error** if file not found
Script Execution	Continues execution after error	Stops execution after error
Use Case	Non-critical file inclusion	Critical file inclusion (e.g., DB connection)

Example of include

```
// header.php
echo "<h1>Welcome to My Website</h1>";

// main.php
```

```
include 'header.php';
echo "This is the main content.";
```

Output:

Welcome to My Website

This is the main content.

If header.php is missing, the script continues running but with a warning.

Example of require

```
// config.php

$db = mysqli_connect("localhost", "root", "", "mydb");



// main.php

require 'config.php';

echo "Database connection established.";
```

If config.php is missing or has errors, the script stops with a **fatal error**.

Functions in PHP

- In PHP, a **function** is a self-contained block of code designed to perform a specific task.
- Functions help in **code reuse**, **modularity**, and **efficient programming** by grouping related statements together under a single name.

Uses :

- Avoid code repetition
- Organize code into reusable blocks
- Make code more readable and maintainable

- ❖ Facilitate debugging and testing
- ❖ Improve program modularity

Syntax of a Function

```
function functionName(parameters) {
   // Code to be executed
}
```

function → keyword to define a function

functionName → name of the function (must start with a letter or underscore)

parameters → optional input values

Calling a Function

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

greet(); // Function call
```

Types of Functions in PHP

1. Built-in Functions

PHP comes with a wide range of predefined functions.

Examples:

```
strlen("hello");       // Returns 5
strtoupper("php");     // Returns "PHP"
date("Y-m-d");         // Returns current date
```

2. User-defined Functions

Functions defined by the programmer for specific tasks.

Example:

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

```
echo add(10, 20); // Outputs 30
```

Function with Parameters

```
function greetUser($name) {
   echo "Hello, $name!";
}
greetUser("Vineela"); // Outputs: Hello, Vineela!
```

Function with Return Value

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

Default Parameter Values

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

Pass by Value and Pass by Reference in PHP

- In PHP, when passing arguments to functions, there are two methods:
  - **Pass by Value**
  - **Pass by Reference**
- Understanding the difference between these two is crucial, especially when working with functions that modify variables.

1. Pass by Value

- In **pass by value**, a **copy** of the variable is passed to the function.
- Changes made inside the function **do not affect** the original variable.

Example: Swapping Two Numbers Using Pass by Value

```
function swap($a, $b) {
   $temp = $a;
   $a = $b;
   $b = $temp;
   echo "Inside function (Pass by Value): A = $a, B = $b<br>";
}

$x = 10;
$y = 20;
swap($x, $y);
echo "Outside function: A = $x, B = $y<br>";
```

Output:

```
Inside function (Pass by Value): A = 20, B = 10
Outside function: A = 10, B = 20
```

- Original values of $x and $y remain **unchanged** outside the function.

2. Pass by Reference

- In **pass by reference**, the **memory address (reference)** of the variable is passed to the function.
- Changes made inside the function **affect the original** variable.

Example: Swapping Two Numbers Using Pass by Reference

```
function swapRef(&$a, &$b) {
   $temp = $a;
   $a = $b;
   $b = $temp;
   echo "Inside function (Pass by Reference): A = $a, B = $b<br>";
}

$x = 10;
$y = 20;
swapRef($x, $y);
echo "Outside function: A = $x, B = $y<br>";
```

Output:

```
Inside function (Pass by Reference): A = 20, B = 10
Outside function: A = 20, B = 10
```

- Original values of `$x` and `$y` are **successfully swapped** outside the function too.

keywords

- Keywords in PHP are reserved words that perform specific functions and are part of the PHP language syntax.
- In PHP, **keywords** (also known as **reserved words**) are predefined words that have special meaning in the language.
- These words are reserved by the PHP compiler and **cannot be used as names for variables, functions, classes, or identifiers**.
- These words are used to perform control flow, declare data types, define functions or classes, handle loops, and more.

Characteristics of PHP Keywords

- They are **case-insensitive** (e.g., `IF`, `if`, and `If` are treated the same).
- They **cannot** be used as variable, function, or class names.
- They form the **core structure** of PHP syntax.

List of keywords in PHP:

abstract	and	array	as
break	callable	case	catch
class	clone	const	continue
declare	default	die	do
echo	else	elseif	empty
enddeclare	endfor	endforeach	endif
endswitch	endwhile	eval	exit
extends	final	finally	for
foreach	function	global	goto

if	implements	include	include_once
instanceof	insteadof	interface	isset
list	namespace	new	or
print	private	protected	public
require	require_once	return	static
switch	throw	trait	try
unset	use	var	while
xor	yield		

Recursion

- **Recursion** is a programming technique where a function calls **itself** either **directly** or **indirectly** to solve a problem.
- In PHP, recursion is a powerful concept used for solving problems that can be broken down into **smaller, similar sub-problems** — such as factorials, Fibonacci series, tree traversal, etc.
- A recursive function must have:
  - **Base Case** – A condition to stop recursion.
  - **Recursive Case** – Function calls itself with a different argument.

Example:

```
<?php

function factorial($n) {
   if ($n == 1) {
      return 1; // base case
   } else {
      return $n * factorial($n - 1); // recursive call
   }
}

echo factorial(3); // Output:6\
?>
```

Dry Run:

factorial(3) → 3 * factorial(2)
→ 3 * (2 * factorial(1))
→ 3 * 2 * 1 = 6

Applications of Recursion

- Calculating factorial
- Generating Fibonacci series
- Searching and sorting algorithms (QuickSort, MergeSort)
- Tree and graph traversal (e.g., DFS)
- Solving mathematical puzzles (e.g., Tower of Hanoi)