

SWDPP401

PHP PROGRAMMING

Apply PHP Programming

Competence

RQF Level: 4

Learning Hours



Credits: 13

Sector: ICT and Multimedia

Trade: Software Development

Module Type Specific

Curriculum: ICTSWD4002: TVET

Certificate IV in Software Development

2024

Level 4 SWD: PHP Programming

LO 1: Preparation of PHP Programming environment

▪ Definition of key terms

✓ PHP

PHP, which stands for "Hypertext Preprocessor," is a popular server-side scripting language designed for web development.

✓ Interpreter

The PHP interpreter is a program or module installed on the web server. Its primary job is to read and execute PHP code. It interprets the code line by line, executing the instructions, and generating dynamic HTML content.

✓ Open Source

Open source refers to software that is distributed with a license that grants users the freedom to view, modify, and distribute the source code of the software. PHP itself is an open-source programming language, which means that its source code is freely available for anyone to examine, modify, and use.

✓ Web Server

A web server is software or hardware that serves as the foundation for hosting and delivering websites, web applications, and other web content over the internet. When a user requests a web page or resource by entering a URL (Uniform Resource Locator) into their web browser, the web server processes that request, retrieves the requested content, and sends it back to the user's browser, allowing them to view the web page or access the resource.

✓ Apache

This is one of the most widely used open-source web server software. It's often referred to simply as "Apache." Apache HTTP Server is used to serve websites and web applications on the World Wide Web. It's known for its stability, flexibility, and extensibility. Many websites, including some of the largest and most popular ones, use Apache as their web server software.

✓ **Database**

This refers to a collection of logically related data. In most cases we use relational databases where data is arranged in tabular format.

✓ **DBMS**

This is an application that define, creates and maintains the database. It includes DBMSs like MySQL, MySQL server, Oracle, etc.

✓ **MySQL**

This is one of the database management systems that uses SQL as a database language for database user interaction.

✓ **Static website**

A website with no user authority to modify its components. Website users visit and use website components without changing any component on the website.

✓ **Dynamic website**

This website allows the user to modify and add components to the existing web contents, like changing colors, giving comments and interacting the website support team.

▪ **Purpose of PHP**

Server-Side Scripting: PHP is primarily used to write server-side scripts. When a user requests a web page, the PHP code on the server processes the request, interacts with databases if necessary, and generates dynamic HTML content to send to the user's web browser.

Dynamic Web Pages: PHP allows web developers to create web pages that can change and adapt in real-time based on user input or other variables. This dynamic nature is essential for building interactive websites, such as social media platforms, ecommerce sites, and content management systems (CMS).

Database Interaction: PHP can connect to various databases, including MySQL, PostgreSQL, SQLite, and others. This enables developers to retrieve and manipulate data from databases, making it possible to build data-driven web applications and websites.

User Authentication and Security: PHP provides features and functions for user authentication and security.

Developers can use PHP to handle user login, session management, and data validation to protect against security vulnerabilities like SQL injection and cross-site scripting (XSS).

Content Management Systems (CMS): Many popular CMS platforms, such as WordPress, Drupal, and Joomla, are built using PHP. PHP allows developers to create customizable and extensible CMS systems that simplify website management for non-technical users.

API Development: PHP can be used to create web-based APIs (Application Programming Interfaces) that allow different software systems to communicate with each other over the internet. This is crucial for building web services and integrating applications.

Command-Line Scripting: Besides web development, PHP can also be used for command-line scripting. This allows developers to automate various tasks and perform system-level operations.

Cross-Platform Compatibility: PHP is compatible with various operating systems (Windows, Linux, macOS) and web servers (Apache, Nginx, IIS), making it a versatile choice for web development.

Open Source and Extensible: PHP is open source, which means it is constantly evolving, and a vast community of developers contributes to its improvement. It has a wide range of libraries, frameworks (e.g., Laravel, Symfony, CodeIgniter), and extensions that can be used to extend its functionality

▪ **Important characteristics of PHP**

Server-Side Scripting: PHP is primarily used for server-side scripting, meaning that it runs on a web server to process requests from clients (usually web browsers).

It generates dynamic web content, such as HTML, XML, JSON, and more, in response to these requests.

Embedded in HTML: PHP code can be embedded directly within HTML documents, making it easy to mix dynamic and static content. PHP scripts are enclosed in special delimiters, typically `<?php` and `?>`.

Database Integration: PHP can connect to various databases, including MySQL, PostgreSQL, Oracle, and others, making it a powerful tool for building database driven web applications.

Extensive Libraries: PHP has a vast standard library and a thriving community that has developed numerous open-source libraries and frameworks, such as Laravel, Symfony, and CodeIgniter, which simplify web development tasks.

Cross-Platform: PHP is compatible with various operating systems, including Windows, Linux, macOS, and more. It can run on a wide range of web servers, including Apache, Nginx, and Microsoft IIS.

High Performance: PHP has improved its performance significantly over the years, especially with the introduction of PHP 7 and later versions, which include features like the Zend Engine and opcode caching to enhance execution speed.

Community Support: PHP has a large and active community of developers who contribute to its development, share code, and offer support through online forums, blogs, and resources.

Versatile Applications: PHP is used for a wide range of web-related tasks, such as creating websites, blogs, content management systems (CMS), e-commerce platforms, and web services.

Open Source: PHP is open-source software, which means it is free to use, modify, and distribute. This has contributed to its widespread adoption and continuous improvement.

▪ **PHP Development Tools**

- ✓ XAMPP
- ✓ WAMP/MAMP/LAMP
- ✓ IDEs /Text Editors
- ✓ Browser

▪ **Installation of XAMPP/WAMP or LAMP**

XAMPP (Cross-Platform Apache, MySQL, PHP, and Perl) is a popular web development environment that allows you to run a local web server on your computer for testing and development purposes. To install XAMPP you follow the following steps.

- ✚ Download XAMPP by Visiting the official XAMPP website
(<https://www.apachefriends.org/index.html>)
- ✚ Install XAMPP by double-clicking on the downloaded files and follow instructions.
- ✚ Open XAMMP Control Panel from installed programs on your computer
- ✚ Open required XAMPP services on XAMPP Control Panel (Apache, MySQL, Perl, etc.)

▪ **Configuration of environment**

✓ **Ports**

To configure ports for XAMPP, you typically need to adjust the settings for the Apache and MySQL servers, as these are the two primary components that use ports. By Clicking on the "Config" button for Apache in the XAMPP Control Panel and select "httpd.conf."

Apache default port:80. This can be changed to available port 8080

MySQL Uses default port 3306, you can change it to listen to any available port.

✓ **Browser**

Examples of common browsers used include

(Chrome, Firefox, Internet explorer, opera, etc.). It is recommended to use chrome browser of latest version of Firefox

✓ **Services**

XAMPP includes several services and components that are essential for web development.

Here are the main services provided by XAMPP:

Apache: XAMPP includes the Apache HTTP Server, one of the most widely used web servers. Apache is responsible for serving web pages and handling HTTP requests.

MySQL: XAMPP bundles the MySQL database management system. MySQL is a popular relational database management system used for storing and managing data in web applications.

PHP: PHP is a server-side scripting language used for web development. XAMPP includes PHP, which allows you to create dynamic web applications and interact with the database.

phpMyAdmin: phpMyAdmin is a web-based graphical user interface (GUI) tool for managing MySQL databases. It is included in XAMPP to facilitate database administration tasks.

FileZilla FTP Server: XAMPP also includes FileZilla FTP Server, which can be used for transferring files to and from your web server. FTP (File Transfer Protocol) is commonly used for uploading website files.

Mercury Mail Server: Mercury is an email server included in XAMPP that can be used for testing email functionality in web applications.

Tomcat: In addition to the traditional LAMP (Linux, Apache, MySQL, PHP) stack, XAMPP also includes Tomcat, which is an application server for running Java-based web applications.

✓ IDEs Extensions

IDEs often support extensions or plugins that enhance their functionality and adapt them to specific development workflows. For example:

Visual Studio Code (VS Code):

Extensions: VS Code has a vast marketplace of extensions that cater to web development, including:

- Live Server: Launches a local development server with live reloading.

- ESLint: Provides real-time linting and code analysis for JavaScript.

- Prettier: An opinionated code formatter for HTML, CSS, JavaScript, and more.

- Debugger for Chrome: Allows debugging JavaScript in Google Chrome.

- GitLens: Enhances Git integration within the IDE.

- React Native Tools: Offers support for React Native development.

Sublime Text:

Packages: Sublime Text uses packages for extending functionality:

Emmet: Provides shortcuts for writing HTML and CSS quickly.

Package Control: Helps manage packages easily.

Babel: JavaScript syntax highlighting and transpilation.

Sublime Linter: Offers real-time code linting.

▪ Application of PHP concepts

✓ PHP file extension

File extension and Tags are used in order for the server to identify our PHP files and scripts, we must save the file with the “. php” extension. Older PHP file extensions include (.phtml, .php3, .php4, .php5, .phps)

PHP was designed to work with HTML, and as such, it can be embedded into the HTML code.

<BODY> {PHP Code} </BOBY>

Note: You can create PHP files without any html tags and that is called Pure PHP file.

The server interprets the PHP code and outputs the results as HTML code to the web browsers.

In order for the server to identify the PHP code from the HTML code, we must always enclose the PHP code in PHP tags.

A PHP tag starts with the less than symbol followed by the question mark and then the words php”. (**<? Php ?>**)

PHP is a case sensitive language, “VAR” is not the same as “var”. The PHP tags themselves are not case-sensitive, but it is strongly recommended that we use lower case letter. The code below illustrates the above point

✓ Syntax

Basic PHP Syntax

A PHP script can be placed anywhere in the document. A PHP script starts with **<? php** and ends with **?>**:

<? php


```
// PHP code goes here
```

```
?>
```

Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<body> <? php echo "My first PHP  
script!";
```

```
?>
```

```
</body>
```

```
</html>
```

Output:

My first PHP script!

✓ **Variable**

Variables are "containers" for storing information. In PHP Creating a variable is called variable declaration. Variable starts with the \$ sign, followed by the name of the variable:

Example

```
<? php
```

```
$txt = "Hello world!";
```

```
$x = 5;
```

```
$y = 10.5;
```

```
?>
```

After the execution of the statements above, the variable \$txt will hold the value Hello world! The variable \$x will hold the value 5, and the variable \$y will hold the value 10.5.

Note:

1. When you assign a text value to a variable, put quotes around the value.
2. Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

1. A variable start with the \$ sign, followed by the name of the variable
2. A variable name must start with a letter or the underscore character
3. A variable name cannot start with a number
4. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
5. Variable names are case-sensitive (\$age and \$AGE are two different variables)

Output Variables

The PHP echo statement is often used to output data to the screen. The following example will show how to output text and a variable:

Example

```
<? php  
$txt = "W3Schools.com"; echo "I love $txt!";  
?>
```

The following example will produce the same output as the example above:

Example `<? php`

```
$txt = "W3Schools.com"; echo "I love “. $txt. “!";  
?>
```

The following example will output the sum of two variables:

Example

```
<? php $x = 5; $y = 4;  
echo $x + $y;  
?>
```

PHP is a Loosely Typed Language. In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value.

- PHP Variables Scope

✓ Operators

Operators are used to perform operations on variables and values. PHP divides the operators in the following groups:

1. Arithmetic operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

2. Assignment operators

The PHP assignment operators are used with numeric values to write a value to a variable. The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

3. Comparison operators

The PHP comparison operators are used to compare two values (number or string):
Like ==(Equal), === (Identical)

4. Increment/Decrement operators

The PHP increment operators are used to increment a variable's value. The PHP decrement operators are used to decrement a variable's value. (-- or ++)

5. Logical operators

The PHP logical operators are used to combine conditional statements. (AND / OR)

6. String operators

PHP has two operators that are specially designed for strings. (Concatenation.)

7. Array operators

The PHP array operators are used to compare arrays (Union, Equality, Identity, etc)

8. Conditional Operator

The ternary operator allows us to simplify some PHP conditional Statements.

Example:

```
<? php
```

```
$a = 10;
```

```
$b = 20;
```

```
/* If condition is true then assign a to result otherwise b */
```

```

$result = ($a > $b)? $a: $b; echo "TEST1: Value of
result is $result<br/>";
/* If condition is true then assign a to result otherwise b */
$result = ($a < $b)? $a: $b; echo "TEST2: Value of
result is $result<br/>"; ?>

```

✓ **Data types**

The values assigned to a PHP variable may be of different data types including simple string and numeric types to more complex data types like arrays and objects. PHP supports total eight primitive data types: Integer, Floating point number or Float, String, Booleans, Array, Object, resource and NULL. These data types are used to construct variables. Now let's discuss each one of them in detail.

✓ **Variable scope**

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used. PHP has three different variable scopes: (Local, Global, Static).

Global and Local Scope

A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example

```

<? php
$x = 5; // global scope function
myTest() {
// using x inside this function will generate an error echo
"<p>Variable x inside function is: $x</p>";
} myTest(); echo "<p>Variable x outside
function is:
$x</p>"; ?>

```

A variable declared within a function has a LOCAL SCOPE and can only be accessed within that function:

Example <? php function myTest() { \$x = 5; // local scope echo "<p>Variable x inside function is: \$x</p>"; } myTest();

// using x outside the function will generate an error echo "<p>Variable x outside function is: \$x</p>";

?>

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

PHP The Global Keyword

The global keyword is used to access a global variable from within a function. To do this, use the global keyword before the variables (inside the function):

Example

```
<? php
$x = 5; $y = 10;
function yTest()
{global $x, $y;
$y = $x + $y;
} myTest(); echo $y; //
outputs 15
?>
```

PHP also stores all global variables in an array called \$GLOBALS[index]. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

Example

```
<? php
$x = 5; $y = 10; function
myTest() {
$GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
} myTest(); echo $y; //
outputs 15
?>
```

PHP The Static Keyword

Normally, when a function is completed/executed, all of its variables are deleted.

However, sometimes we want a local variable NOT to be deleted. We need it for a further job. To do this, use the static keyword when you first declare the variable:

Example

```
<? php function myTest() { static $x = 0; echo $x;  
$x++;  
} myTest();  
myTest(); myTest();  
?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

✓ Constants

Constants in PHP are a way to store and manage values that remain the same throughout the execution of a script. These values are immutable, meaning they cannot be changed once they are defined. PHP constants are often used to store configuration settings, fixed values, or any data that should not be altered during the execution of a script.

Advantages of Constants

01. Constants provide a way to store and manage configuration settings in a central location.
02. They help make your code more readable by providing meaningful names for values.
03. Constants cannot be accidentally modified, making your code more robust.

Rules for Constant Names

01. Constant names are case-sensitive.
02. By convention, constant names are typically written in uppercase.
03. The name can consist of letters, numbers, and underscores but cannot start with a number.

Defining Constants

You can define constants in PHP using the **define ()** function or the **const** keyword. The general syntax for both methods is:

Using define ()

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

Using const

```
const CONSTANT_NAME = "constant_value";
```

Note that you have to replace `CONSTANT_NAME` with the name of your constant and `"constant_value"` with the value you want to assign to it.

Accessing Constants

You can access constants anywhere in your PHP script, and they are accessible globally.

To access a constant, you simply use its name, without the dollar sign (\$).

For example:

```
echo CONSTANT_NAME;
```

Magic Constants

PHP also provides a set of predefined constants known as "magic constants" that change based on the context in which they are used. Some commonly used magic constants include `__FILE__`, `__LINE__`, and `__DIR__`.

For example:

```
echo __FILE__; // Displays the current file's path and name  
echo __LINE__; // Displays the current line number
```

✓ Comment

In PHP, Comments are lines of codes which are written to explain the functionality of the source code but does not affect the execution of the code.

Types of PHP comments

1. Single-line comments:

Single-line comments are used to comment out a single line of code or provide a brief explanation on the same line.

```
// This is a single-line comment using double slashes
```

OR

```
This is a single-line comment using a hash symbol
```

Example:

```
$variable = 42; // Assigning the value 42 to the variable
```

2. Multi-line comments:

Multi-line comments allow you to comment out multiple lines of code or provide detailed explanations spanning several lines.

Example

```
/* This is a multi-line comment  
using the forward slash and asterisk */
```

Practical Example:

```
/*  
This function calculates the sum of two numbers.  
$a - the first number  
$b - the second number  
*/  
function add ($a, $b) {  
return $a + $b;  
}
```

Comments are essential for making your code more understandable for yourself and others who might work on or read your code. They can also help you debug and troubleshoot your code by providing context and explanations.

✓ Date and time

PHP provides a wide range of functions and capabilities for working with dates and times. This is important for web development and other applications that need to handle time-related data. PHP's date and time functions allow you to perform various operations such as displaying the current date and time, formatting dates, parsing date strings, performing calculations, and more.

How to work with dates and times in PHP:

Current Date and Time

You can easily obtain the current date and time using the `date()` function.

For example:

```
echo date("Y-m-d H: i: s"); // Displays the current date and time in "Y-m-d H: i: s" format.
```

Formatting Dates:

PHP provides various formatting options to display dates and times in a way that suits your needs. For instance:

```
echo date("F j, Y, g:i a"); // Displays a date like "October 11, 2023, 2:30 pm".
```

Creating Date Objects

You can create date objects using the `DateTime` class to perform date calculations and manipulations. For example:

```
$date = new DateTime("2023-10-11");  
$date->modify("+1 day"); echo $date->format("Y-m-d"); // Displays  
"2023-10-12".
```

Time zone Handling

When working with date and time in different time zones, you can use the `DateTimeZone` class to handle time zone conversions.

```
$date = new DateTime("2023-10-11", new  
DateTimeZone("America/New_York"));  
$date->setTimezone(new DateTimeZone("UTC")); echo $date->format("Y-m-d H:i:s");
```

Date Arithmetic

PHP allows you to perform date arithmetic, such as adding or subtracting days, months, or years from a given date.

```
$date = new DateTime("2023-10-11");  
$date->add(new DateInterval("P7D")); // Add 7 days  echo $date-  
>format("Y-m-d"); // Displays "2023-10-18".
```

Parsing Date Strings:

You can convert date strings to DateTime objects using the DateTime::createFromFormat() method.

```
$dateString = "2023-10-11";  
$date = DateTime::createFromFormat("Y-m-d", $dateString);  echo $date-  
>format("F j, Y");
```

Timestamps:

PHP can work with timestamps, which are the number of seconds since January 1, 1970 (Unix epoch). You can use time() to get the current timestamp and date() or DateTime to format it.

```
$timestamp = time();  
echo date("Y-m-d H:i:s", $timestamp);
```

Date and Time Functions

PHP provides a wide range of date and time functions, such as strtotime(), strftime(), date_diff(), and more, for performing various operations on dates and times.

✓ String concatenation

In PHP, you can concatenate strings using the . (dot) operator. This operator allows you to combine two or more strings into a single string.

Example 1

```
$string1 = "Hello, ";  
$string2 = "world!";  
$combinedString = $string1. $string2; echo $combinedString;
```

In the above example, \$combinedString will contain "Hello, world!" after concatenation.

Example 2

You can also use the . = operator to concatenate a string to an existing string variable:

```
$greeting = "Hello, "; $greeting. =  
"world!";  
echo $greeting;
```

This will also result in the output "Hello, world!".

Example 3

You can concatenate as many strings as you need using the . operator, and you can mix variables and string literals in the concatenation. Here's another example:

```
$name = "John";  
$greeting = "Hello, “. $name. "! Welcome to our website."; echo $greeting;
```

In this case, the output will be "Hello, John! Welcome to our website.".

✓ Condition statement

Conditional statements are PHP statements used to make decisions and execute different blocks of code based on whether certain conditions are met. These include **if...**, **if... else**, **if... else if... else**, **Ternary Operators** and **Switch** statements.

Advantages of Conditional statement

1. Helps to control the flow of your PHP code
2. Makes a php program more dynamic and responsive to different situations.
3. They enable you to create flexible and interactive web applications and perform actions based on user input or data conditions.

If Statement

The if statement is used to execute a block of code only if a specified condition is true. If the condition evaluates to true, the code within the if block is executed; otherwise, it's skipped. **Example**

```
$age = 25; if ($age >= 18) { echo "You are an adult.";  
}
```

if-else Statement

The if-else statement is an extension of the if statement. It allows you to execute one block of code when a condition is true and another block when it's false.

Example

```
$age = 15; if ($age >=
18) {      echo "You are an
adult.";
} else {      echo "You are not
an adult.";
}
```

if-elseif-else Statement

This is used when you have multiple conditions to check. You can use multiple elseif blocks to check different conditions, and if none of them is true, you can have a final else block.

Example

```
$score = 85; if
($score >= 90) {
echo "A";
} elseif ($score >= 80) {
echo "B";
} elseif ($score >= 70)
{      echo "C"; } else {
echo "F";
}
```

switch Statement

The switch statement is another way to handle multiple conditions, especially when you're comparing a single variable against multiple possible values. It's often more concise than a series of if statements.

Example

```
$day = "Monday";  
switch ($day) {  
case "Monday":  
    echo "It's the start of the  
week.";    break;    case  
"Friday":  
    echo "It's almost the  
weekend.";    break;    default:  
    echo "It's some other day."  
}
```

Ternary Operator

The ternary operator (?:) provides a shorthand way to write simple if-else conditions in a single line. It's often used for assigning values based on a condition.

Example

```
$age = 20;  
$status = ($age >= 18)? "Adult": "Minor";    echo $status;
```

✓ Arrays

An array is a versatile and fundamental data structure used to store and manage collections of data. It allows you to group multiple values together under a single variable name, making it easier to work with and manipulate data in your programs.

PHP arrays can hold various types of data, such as numbers, strings, objects, and even other arrays.

They are essential for tasks like storing lists of items, iterating through data, and organizing information.

Types of Arrays

Indexed Arrays

The most basic type of PHP array is the indexed array.

Elements in an indexed array are assigned numeric keys (indexes), starting from 0.

You can create an indexed array using square brackets or the array () function.

Example

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

Associative Arrays

In an associative array, elements are assigned specific keys that you define, making it easy to access values using those keys.

You create associative arrays using the => operator.

Example

```
$person = [  
    "first_name" => "John",  
    "last_name" => "Doe",  
    "age" => 30  
];
```

Multidimensional Arrays

PHP supports multidimensional arrays, where each element can be another array.

This is useful for storing structured data.

Example:

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

Accessing Array Elements

You can access array elements by specifying the array name followed by the index or key in square brackets.

Example

```
$colors = ["red", "green", "blue"];  
echo $colors[1]; // Outputs "green"  
  
$person = [  
    "first_name" => "John",  
    "last_name" => "Doe",  
    "age" => 30  
];  
echo $person["first_name"]; // Outputs "John"
```

Modifying Array Elements

You can change the value of an element by referencing its key or index and assigning a new value.

Example:

```
$colors[2] = "yellow"; // Changes the third element to "yellow"  
$person["age"] = 31; // Updates the age to 31
```

✓ Loop

Loops are control structures that allow you to repeatedly execute a block of code. They are essential for performing repetitive tasks, such as iterating through arrays, processing data, or performing actions until a certain condition is met. **Examples of PHP Loops for Loop**

The for loop is used when you know the number of times you want to execute a block of code. It has three parts: initialization, condition, and increment.

Example:

```
for ($i = 0; $i < 5; $i++) {  
    echo "Iteration $i<br>";  
}
```

while Loop

The while loop continues to execute a block of code as long as a specified condition is true.

Example

```
$i = 0; while ($i < 5) {    echo "Iteration $i<br>";  
    $i++;  
}
```

do-while Loop

The do-while loop is similar to the while loop, but it executes the code block at least once before checking the condition.

Example

```
$i = 0; do {    echo "Iteration $i<br>";  
    $i++;  
} while ($i < 5);
```

foreach Loop

The foreach loop is used for iterating over elements in an array or objects. It simplifies the process of iterating through arrays.

Example

```
$colors = array("red", "green", "blue"); foreach ($colors as  
$color) {    echo "Color: $color<br>";  
}
```


foreach Loop with Associative Arrays

You can also use foreach with associative arrays to iterate over key-value pairs.

Example

```
$person = array("first_name" => "John", "last_name" => "Doe", "age" => 30);  
foreach ($person as $key => $value) {    echo "$key: $value<br>";  
}
```

✓ Function

Introduction to function

A function is a block of code that performs a specific task or set of tasks. Functions are essential for organizing and reusing code, making your PHP programs more modular and maintainable. They allow you to encapsulate a piece of functionality that can be called and executed whenever needed. Functions can accept input (parameters) and can return output (a value)

Types of Function

- System defined/library/inbuilt
- User defined

Advantages of Function

- A function is created once but used many times, often from more than one program.
- It reduces duplication within a program.
- Debugging and testing a program becomes easier when the program is subdividing.

1. Built-in functions

String Manipulating Functions:

strlen()	Returns the length of a string.
str_replace()	Replaces all occurrences of a search string with a replacement string.
substr()	Returns a portion of a string.
strtolower()	Converts a string to lowercase.
strtoupper()	Converts a string to uppercase.

Array Functions:

array_push()	Adds elements to the end of an array.
array_pop()	Removes and returns the last element of an array.
array_merge()	Merges two or more arrays.
array_key_exists()	Checks if a key exists in an array.
array_search()	Searches for a value in an array and returns its key.

Math Functions:

abs()	Returns the absolute value of a number.
round()	Rounds a floating-point number to the nearest integer.
rand()	Generates a random integer.
sqrt()	Returns the square root of a number.

Date and Time Functions:

date()	Formats a timestamp as a date and time.
time()	Returns the current Unix timestamp.
strtotime()	Parses a date and time string and returns a Unix timestamp.
strftime()	Formats a local time and/or date.

File System Functions:

file_get_contents()	Reads a file into a string.
file_put_contents()	Writes a string to a file.
unlink()	Deletes a file.
file_exists()	Checks if a file or directory exists.
is_file()	Checks if a path is a regular file.

Database Functions:

mysqli_connect()	Establishes a connection to a MySQL database.
mysqli_query()	Executes an SQL query on a database.

mysqli_fetch_array()	Fetches a result row as an associative or numeric array.
mysqli_close()	Closes the connection to the MySQL server.

Regular Expression Functions:

preg_match()	Performs a regular expression match.
preg_replace()	Performs a regular expression search and replace.
preg_match_all()	Performs a global regular expression match.

HTTP Functions:

file_get_contents()	Retrieves the contents of a file or a URL.
curl_init()	Initializes a cURL session for making HTTP requests.
http_response_code()	Gets or sets the HTTP response code.

JSON Functions:

json_encode()	Converts a PHP variable to a JSON string.
json_decode()	Converts a JSON string to a PHP variable.

2. User-defined functions

User-defined function is a block of code that performs a specific task and can be defined by the programmer. It allows you to group code into a reusable module, making your code more modular and easier to maintain.

How to define Function

A function will be executed by a call to the function.

Syntax function function_name()

```
{
    code to be executed;
}
```

Example

Consider a simple function that shows my name when it is called.

```
<? php function writeName ()
{
    echo "My Name";
}
```

```
writeName ();
```

```
?>
```

Output: My Name

In the above example first define a function () writeName. inside function body print "My Name" string using echo statement. To call a function always only function name is required.

✓ **Calling function**

To call a function in PHP, you simply use the function name followed by parentheses.

If the function requires any parameters, you'd put them inside the parentheses.

Example

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

```
// Calling the function greet("John");
```

In the above example, we have a function called greet that takes a parameter \$name.

When we call the function with greet("John"), it echoes "Hello, John!"

✓ **Function recursion**

Function recursion in PHP is when a function calls itself during its execution. This can be a powerful technique for solving problems that can be broken down into smaller, similar sub problems.

Example:

In example below, the factorial function calculates the factorial of a number using recursion.

The base case (\$n <= 1) is the stopping condition to prevent an infinite loop. If the base case is not met, the function calls itself with a smaller argument (\$n - 1).

```
<? php function factorial($n) {  
    if ($n <= 1) {  
        return 1;  
    } else {  
        return $n * factorial ($n - 1);  
    }  
}
```

```
// Example usage
$result = factorial (5);
echo "Factorial of 5 is: $result";
?>
```

✓ **Super Global variables**

Super global variables refer to a group of predefined global arrays that are accessible from any part of your script. These variables are used to collect data from forms, cookies, server variables, and more.

Examples

1. **\$GLOBALS**: This is a global associative array that contains references to all variables that are currently defined in the global scope of the script.
2. **\$_SERVER**: This array holds information about the server and the execution environment. It includes information such as headers, paths, and script locations.
3. **\$_GET**: This array is used to collect form data after submitting an HTML form with the GET method. It's also good for passing data via URLs.
4. **\$_POST**: Similar to **\$_GET**, but used for collecting form data after submitting an HTML form with the POST method. It's more secure for sensitive data.
5. **\$_REQUEST**: This array contains data from both **\$_GET**, **\$_POST**, and **\$_COOKIE**. It's a way to access data sent to the script regardless of the HTTP method used.
6. **\$_SESSION**: Used to store session variables across multiple pages. Session variables are used to store individual client's information on the server for later use.
7. **\$_COOKIE**: This array holds data sent to the server by the client's browser in the form of cookies.
8. **FILES**: Used to collect information about uploaded files via HTTP POST. It's often used in conjunction with the HTML file input element.
9. **\$_ENV**: An associative array containing environment variables.
10. **\$_SERVER['REQUEST_METHOD']**: Returns the request method used to access the page (e.g., 'GET', 'POST').

✓ **PHP file handling**

PHP allows you to perform various file operations, including opening, reading, writing, closing, and deleting files.

1. Opening a File:

To open a file in PHP, you can use the `fopen()` function. It takes two arguments: the file name (including the path) and the mode in which you want to open the file.

Common modes include "r" for reading, "w" for writing (truncating the file if it already exists), "a" for appending, and more. `$file = fopen("example.txt", "r"); //`

Open for reading

2. Reading a File:

You can read the contents of a file using functions like `fread()` or `fgets()` in a loop to read line by line.

Example using fread():

```
$file = fopen("welcome.txt", "r");  
if ($file) { while (!feof($file)) {  
    $data = fread($file, 1024);  
    echo $data;  
}  
fclose($file); // Close the file when done }
```

3. Writing to a File:

To write to a file, you can open it in write mode ("w") or append mode ("a").

Use the `fwrite()` function to write data to the file:

```
$file = fopen("welcome.txt", "w"); // Open for writing (truncates the file) if  
($file) { fwrite($file, "Hello, World!");  
fclose($file); // Close the file when done  
}
```

If you want to append data to an existing file, use "a" as the mode instead of "w".

4. Closing a File:

Always remember to close the file using the `fclose()` function when you're done with it to free up system resources.

```
fclose($file);
```

5. Deleting a File:

To delete a file, you can use the `unlink ()` function. It takes the file name (including the path) as its argument. `$fileToDelete = "example.txt";` if

```
(file_exists($fileToDelete)) {  
    unlink($fileToDelete);  
    echo "File deleted successfully."  
} else {  
    echo "File does not exist."  
}
```

Note: Remember to check if the file exists before attempting to delete it to avoid errors.

▪ Application of PHP Security concepts

✓ PHP form handling

You can handle form submissions using the POST and GET methods. These methods allow you to send data from an HTML form to a PHP script for processing. In examples below, the form data is submitted to a PHP script (`process_post.php` or `process_get.php`) for processing. The PHP script retrieves the form data using the `$_POST` or `$_GET` superglobal arrays and performs the desired actions. Make sure to sanitize and validate the data to ensure security and prevent common web vulnerabilities like SQL injection or cross-site scripting (XSS).

1. Handling a form submission using the POST method:

When you use the POST method, form data is sent in the HTTP request body, making it more suitable for sensitive or large amounts of data. To handle a form submission with the POST method in PHP, follow these steps:

HTML Form (form_post.html):

```
<!DOCTYPE html>
<html>
<head>
  <title>POST Form</title>
</head>
<body>
  <form action="process_post.php" method="post">
    <label for="name">Name:</label>
    <input type="text" name="name" id="name">
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

PHP Script to Process the POST Data (process_post.php):

```
<? php if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST["name"];    //
    Process the form data
    echo "Hello, “. htmlspecialchars($name). "!";
}
?>
```

2. Handling a form submission using the GET method:

When you use the GET method, form data is appended to the URL as query parameters. GET is suitable for simple data and when you want the form data to be visible in the URL. To handle a form submission with the GET method in PHP, follow these steps:

HTML Form (form_get.html):

```
<!DOCTYPE html>
<html>
<head>
  <title>GET Form</title>
</head>
<body>
  <form action="process_get.php" method="get">
    <label for="name">Name:</label>
    <input type="text" name="name" id="name">
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

PHP Script to Process the GET Data (process_get.php):

```
<?php if ($_SERVER["REQUEST_METHOD"] == "GET") {
if (isset($_GET["name"])) {      $name = $_GET["name"];
// Process the form data
    echo "Hello, “. htmlspecialchars($name). "!";
  }
}
?>
```

✓ Validation

Proper validation of form data is important to protect your form from hackers and spammers.

Consider the for below:

PHP Form Validation Example

*** required field**

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other *

Your Input:

The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea.

The HTML code looks like this:

Name: `<input type="text" name="name">`

E-mail: `<input type="text" name="email">`

Website: `<input type="text" name="website">`

Comment: `<textarea name="comment" rows="5" cols="40"></textarea>`

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

Gender:

`<input type="radio" name="gender" value="female">Female`

`<input type="radio" name="gender" value="male">Male`

`<input type="radio" name="gender" value="other">Other`

The Form Element

The HTML code of the form looks like this:

`<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">` When

the form is submitted, the form data is sent with `method="post"`.

What is the `$_SERVER["PHP_SELF"]` variable?

The `$_SERVER["PHP_SELF"]` is a super global variable that returns the filename of the currently executing script.

So, the `$_SERVER["PHP_SELF"]` sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the htmlspecialchars () function?

The htmlspecialchars () function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or JavaScript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The \$_SERVER["PHP_SELF"] variable can be used by hackers. If PHP_SELF is used in your page, then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like

"http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

However, consider that a user enters the following URL in the address bar:

http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that any JavaScript code can be added inside the `<script>` tag! A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How to Avoid `$_SERVER["PHP_SELF"]` Exploits?

`$_SERVER["PHP_SELF"]` exploits can be avoided by using the `htmlspecialchars ()` function.

The form code should look like this:

```
<form method="post" action="<? php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The `htmlspecialchars ()` function converts special characters to HTML entities. Now if the user tries to exploit the `PHP_SELF` variable, it will result in the following output:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script &gt;">
```

The exploit attempt fails, and no harm is done!

Validate Form Data with PHP

The first thing we will do is to pass all variables through PHP's `htmlspecialchars()` function. When we use the `htmlspecialchars()` function; then if a user tries to submit the following in a text field:

```
<script>location.href('http://www.hacked.com')</script>
```

This would not be executed, because it would be saved as HTML escaped code, like this:

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```

 The code is

now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP `trim()` function)

2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test_input().

Now, we can check each \$_POST variable with the test_input() function, and the script looks like this:

```
<!DOCTYPE HTML>
<html>
<head> </head>
<body>
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = test_input($_POST["name"]);
    $email = test_input($_POST["email"]);
    $website = test_input($_POST["website"]);
    $comment = test_input($_POST["comment"]);
    $gender = test_input($_POST["gender"]);
}
function test_input($data)
{
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>
```

<h2>PHP Form Validation Example</h2>

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

```
Name: <input type="text" name="name">
```

```
<br><br>
```

```
E-mail: <input type="text" name="email">
```

```
<br><br>
```

```
Website: <input type="text" name="website">
```

```
<br><br>
```

```
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

```
<br><br> Gender:
```

```
<input type="radio" name="gender" value="female">Female
```

```
<input type="radio" name="gender" value="male">Male
```

```
<input type="radio" name="gender" value="other">Other
```

```
<br><br>
```

```
<input type="submit" name="submit" value="Submit">
```

```
</form>
```

```
<?php echo "<h2>Your Input:</h2>";
```

```
echo $name; echo "<br>"; echo
```

```
$email; echo "<br>"; echo $website;
```

```
echo "<br>"; echo $comment; echo
```

```
"<br>"; echo $gender;
```

```
?>
```

```
</body>
```

```
</html>
```

✓ Cookies and Session

PHP Sessions

A session is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the user's computer.

Start a PHP Session

A session is started with the `session_start()` function. Session variables are set with the PHP global variable: `$_SESSION`.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

```
<?php
// Start the session session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favourite"] = "cat"; echo
"Session variables are set.";
?>
</body>
</html>
```

Note: The `session_start()` function must be the very first thing in your document. Before any HTML tags.

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php"). Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`).

Also notice that all session variable values are stored in the global `$_SESSION` variable:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Echo session variables that were set on previous page echo
"Favorite color is " . $_SESSION["favcolor"] . "<br>"; echo
"Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
</body>
</html>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// remove all session variables
session_unset(); // destroy the
session session_destroy();
?>
</body>
</html>
```

PHP Cookies

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies with PHP

A cookie is created with the `setcookie()` function. **Syntax**

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

Only the name parameter is required. All other parameters are optional

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable `$_COOKIE`). We also use the `isset()` function to find out if the cookie is set:

```
<?php
$cookie_name = "user"; $cookie_value = "John Doe"; setcookie($cookie_name,
$cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day ?>
<html>
<body> <?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else { echo "Cookie '" . $cookie_name . "' is
set!<br>"; echo "Value is: " .
$_COOKIE[$cookie_name];
}
?>
```

```
</body>
</html>
```

Note: The setcookie() function must appear BEFORE the <html> tag.

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead). **Modify a Cookie Value**

To modify a cookie, just set (again) the cookie using the setcookie() function:

```
<?php
$cookie_name = "user"; $cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>
```

```
<?php if(!isset($_COOKIE[$cookie_name])) { echo "Cookie
named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>"; echo
"Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

```
<?php
// set the expiration date to one hour ago setcookie("user", "",
time() - 3600);
?>
```

```

<html>
<body> <?php echo "Cookie 'user' is
deleted.";
?>
</body>
</html>

```

Check if Cookies Are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the `setcookie()` function, then count the `$_COOKIE` array variable:

```

<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body> <?php
if(count($_COOKIE) > 0) { echo
"Cookies are enabled.";
} else {
echo "Cookies are disabled.";
}
?>
</body> </html>

```

▪ Implementation of Object-oriented programming (OOP) in PHP

✓ Definition

Object-Oriented Programming or simply OOPs is a programming approach or paradigm that gives its prime consideration to the data and its associated functions. It uses the concept of wrapping up the data as an object-entity having its associated properties, to provide higher security and less exposure to the data.

✓ **Classes**

The class is the blueprint that is used to hold the objects along with their behavior and properties. In PHP, the class name should be the same as that of the file name with which it has saved the program. A class is a user-defined data type that consists of two entities: Data Members and Member Functions.

Syntax to Define a Class:

```
<?php class Class_Name {  
    }  
?>
```

As you can see above, class_name is the name of the class and it should be the same as that of the file with which you have saved your code.

Data Members: Data Members are the variables that can be of the data type var in PHP.

Data Members act as the data for the source code with which you can meddle around. Data members can have three types of visibility modes that decide the access permission of these members. These modes are private, protected, and public. **Member Functions:** Those data members that have visibility mode as private, and cannot be accessed directly by the class objects. In such cases, member functions come into play. Those functions that are specifically created to access private data members are known as member functions.

Syntax for Declaring Data Members and Member Functions:

// Syntax to define PHP class

```
<?php class Class_Name {  
    // Syntax for declaring data members    var $data_member_name;    // Syntax  
for member function function member_function_name ($argument_name_1,  
$argument_name_2) {  
    [..]  
    }  
    [..]  
    }  
?>
```

Description of the Parameters:

class: The keyword class that needs to be given at the start while defining a class. Just like some other programming languages such as C++, Java, and JavaScript. **Class_Name:** This is the name of the class that is provided after the keyword class and should be the name of the file. For eg: for a file saved as myClass.php, the class name should also be myClass.

data_member_name: This is the name of the data member of your class. You can declare any number of data members in your class, using the keyword var.

member_function_name: This is the name of the member function of your class, you can define a function in your class by using the keyword function before the name of the function.

argument_name: This is the argument of the function that is given while declaring it.

Note: The syntax for defining member functions in a class in PHP is similar to the usual functions that you define in PHP.

The following example will illustrate classes in PHP.

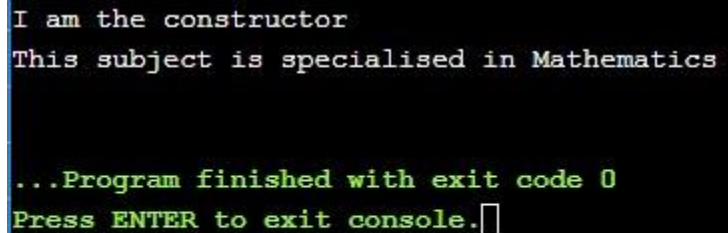
Consider there is a subject, which can have a specialized field as its property. The following program defines a class named subject which has a constructor and a member function that prints the name of the specialized field of that subject.

```
<?php
// define a class named subject class
subject
{
    // defining the constructor of the class    public
    function __construct() {    echo 'I am the
    constructor' .PHP_EOL;
    }
    // defining a member function named
    // field_subject
    // with one argument named field    public function
    subject_field($field) {
    echo 'This subject is specialised in ' . $field .PHP_EOL;
```

```

    }
}
// declare an object of the class
// named obj1 using the "new" keyword.
// as soon as an object is created,
// the default constructor is automatically called.
$obj1 = new subject; // here the default constructor of
// the class subject is called //
automatically.
// calling the function subject_field
// by passing it "Mathematics" as an argument
$obj1->subject_field("Mathematics");
?>

```



```

I am the constructor
This subject is specialised in Mathematics

...Program finished with exit code 0
Press ENTER to exit console.

```

Calling Member Functions in PHP

The objects of a class can be used to access the member function of that class. You have to use the “arrow” (i.e. “->”) operator to achieve this.

Syntax to Access Member Functions and Data Members of a Class:

```

// define a class class class_name
{ // declare data members    var
    $data_member;

    // a member function of the class    public
    function function_name() {
        // define the function here
    }
}

```

```

}
// creating objects of the class // named
class_name using the // "new"
keyword.
$object_name_1 = new class_name;
$object_name_2 = new class_name;
// access member function using //
"arrow" operator.
$object_name_1->function_name();
$object_name_2->function_name();

```

The following example will illustrate how to call member functions in PHP.

```

<?php
class Furniture {
    // declare member variables
    var $name;    var $cost;

    // define the member functions    function
    Name($name){
        $this->name = $name;
    }
    function printName(){    echo $this-
>name .PHP_EOL;
    }
    function Cost($cost){
        $this->cost = $cost;
    }
    function printCost(){    echo $this-
>cost .PHP_EOL;
    }
}

```



```

// create objects for class "Furniture"
$Table = new Furniture();
$Sofa = new Furniture();
$Cupboard = new Furniture();
// call member functions
// by passing string arguments to them.
$Table->Name( "Table" );
$Sofa->Name( "Sofa" );
$Cupboard->Name( "Cupboard" );

// call functions by passing
// integer costs as arguments to them.
$Table->Cost( 7000 );
$Sofa->Cost( 55000 );
$Cupboard->Cost( 25000 );
// call functions to
// print the name of the furniture.
$Table->printName();
$Sofa->printName();
$Cupboard->printName();
// call functions to
// print the cost of the furniture.
$Table->printCost();
$Sofa->printCost();
$Cupboard->printCost();
?>

```

```
Table
Sofa
Cupboard
7000
55000
25000

...Program finished with exit code 0
Press ENTER to exit console.□
```

✓ Objects

An object in a class is an instance that has its own behavior and property. Objects can be related to the entities in real life. It considers everything around you as an object with some of its attributes. For eg: a spaceship is an object. It has properties like fuel, color, speed, material, etc. and it can have functions like `launching_in_space`, `landing_on_a_planet`, etc.

Creating Objects in PHP

Class is one of the most critical OOPs Concepts in PHP. A class can have any number of instances or objects. All the objects of a class will have access to all the data members and member functions of that class. To create an object of a class in PHP, the `new` keyword is used.

The Syntax for Creating an Object in PHP:

```
// define a class class
class_name {
    // declare data members
    // and define member functions...
}
// creating objects of the class
// named class_name using the
// "new" keyword.
$object_name_1 = new class_name;
```

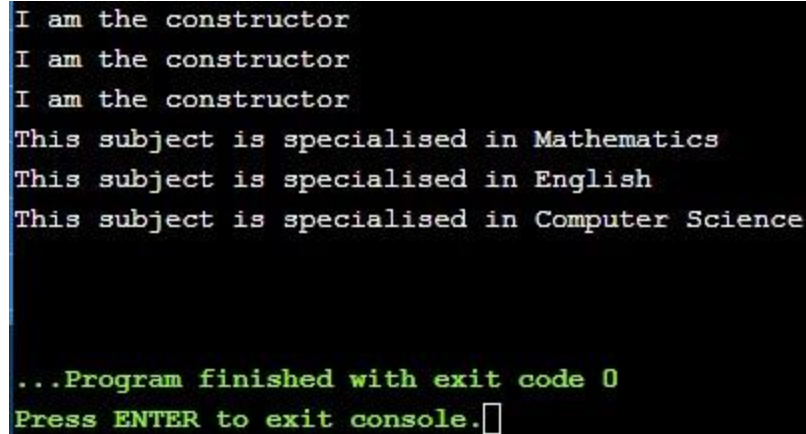
```
$object_name_1 = new class_name;
```

After creating an object, you can use it to call the member functions, or even assign some value to the data members of that class.

The following example will illustrate how to call member functions in PHP. Consider the same example discussed earlier. There is a subject, which can have a specialized field as its property. The following program defines a class named subject which has a constructor and a member function that prints the name of the specialized field of that subject. You can create different objects for the class subject.

```
<?php
// define a class named subject class
subject
{
    // defining the constructor of the class    public
    function __construct() {        echo 'I am the
    constructor' .PHP_EOL;
    }
    // defining a member function named
    // field_subject
    // with one argument named field    public function
    subject_field($field) {        echo 'This subject is specialised in ' .
    $field .PHP_EOL;
    }
}
// declare an objects of the class // using
the "new" keyword.
$obj1 = new subject;    / here the default constructor /
$obj2 = new subject;    / will be called for /
$obj3 = new subject;    / every object created. /
// calling the function subject_field // by passing
it "Mathematics", "English",
// and "Computer Science" as the arguments.
```

```
$obj1->subject_field("Mathematics");  
$obj2->subject_field("English");  
$obj3->subject_field("Computer Science");  
?>
```



```
I am the constructor  
I am the constructor  
I am the constructor  
This subject is specialised in Mathematics  
This subject is specialised in English  
This subject is specialised in Computer Science  
  
...Program finished with exit code 0  
Press ENTER to exit console.█
```

✓ Inheritance

Inheritance in PHP is one of the vital OOPs concepts in PHP, and cannot be overlooked.

Understanding inheritance is critical for understanding the whole point behind object-

oriented programming. For instance, you are a human. Humans inherit from the class

‘Humans’ with characteristic features, such as walking, sitting, running, eating, and so on.

The class ‘Humans’ inherits these characteristic features from the class ‘Mammal’ which

makes the ‘Human’ class a derived class of

‘Mammal’. This ‘Mammal’ class inherits its characteristic features from another class

‘Animal’ which makes the ‘Mammal’ class a derived class of the class ‘Animal’ and makes

the ‘Animal’ a base class.

One of the most astonishing features of inheritance is code reusability. This reusability also

provides you with clean code, and the replication of code gets reduced to almost zero.

Reusing existing codes serves various advantages. It saves time, money, effort, and increases a program’s reliability.

The syntax for declaring a base class to achieve inheritance in PHP:

```
class derived_class_name extends base_class_name {  
    // define member functions of    // the  
    derived class here.  
}
```

The keyword extends is used to define a derived or child class in PHP.

Inheritance in PHP is of 3 types, which you will look at in this article, later on.

- **Single Inheritance:** Single inheritance is the most basic type of inheritance. In single inheritance, there is only one base class and one sub or derived class. The subclass is directly inherited from the base class.

The following example will illustrate single inheritance in PHP.

```
<?php  
// base class named "Furniture" class  
Furniture {    var $cost = 1000;  
    public function printName($name) {        echo 'Class is: Furniture & name of  
furniture is: ' . $name . PHP_EOL;  
    }  
}  
// derived class named "Sofa" class Sofa extends Furniture {    public  
function printName($name) {        echo 'Class is: Sofa & name of furniture  
is: ' . $name . PHP_EOL;  
    // this class can access  
    // data member of its parent class.  
    echo 'Price is: ' . $this->cost . PHP_EOL;  
    }  
}  
$f = new Furniture();  
$s = new Sofa();  
$f->printName('Table');  
$s->printName('Sofa');  
?>
```

```
Class is: Furniture & name of furniture is: Table
Class is: Sofa & name of furniture is: Sofa
Price is: 1000

...Program finished with exit code 0
Press ENTER to exit console.□
```

- **Hierarchical Inheritance:** As the name suggests, hierarchical inheritance shows a treelike structure. There are many derived classes that are directly inherited from a base class.

The following example will illustrate hierarchical inheritance in PHP.

```
<?php
// base class named "Furniture" class Furniture {
public function Furniture() {      echo 'This class is
Furniture ' . PHP_EOL;
    }
}

// derived class named "Sofa" class Sofa
extends Furniture {
}

// derived class named "Sofa" class
Cupboard extends Furniture {
}

// creating objects of
// derived classes
// "Sofa" and "Cupboard"
$s = new Sofa();
$c = new Cupboard();
?>
```

```
This class is Furniture
This class is Furniture

...Program finished with exit code 0
Press ENTER to exit console.█
```

- **Multilevel Inheritance:** Multilevel Inheritance is the fourth type of inheritance that can be found in PHP. Multilevel inheritance can also be explained by a family tree. One base class exists and it inherits multiple subclasses. These subclasses (not every subclass necessarily) acts as base class and further inherits subclasses. This is just like a family having descendants over generations.

The following example will illustrate hierarchical inheritance in PHP.

```
<?php

// base class named "Furniture"

class Furniture {

    public function totalCost() {

        return ' total furniture cost: 60000';

    }

}

// derived class named "Table"

// inherited form class "Furniture"

class Table extends Furniture {

    public function tableCost() {

        return ' table cost: 45000';

    }

}
```

```

// derived class named "Study_Table"

// inherited from class "Table"

class Study_Table extends Table {

    public function studyTableCost() {

        return ' study table cost: 60000';

    }

    public function priceList() {

        echo '1. ' . $this->totalCost() . PHP_EOL;

        echo '2. ' . $this->tableCost() . PHP_EOL;

        echo '3. ' . $this->studyTableCost() . PHP_EOL;

    }

}

// creating object of
// the derived class

$obj = new Study_Table();
$obj->priceList();
?>

```

Inheritance is a splendid OOPs concept in PHP. Inheritance in PHP serves many advantages. There are several reasons why inheritance was introduced in OOPs. You will be exploring some of the major reasons behind the introduction of inheritance in PHP in this section:

One of the most astonishing reasons is that inheritance increases the relatability of the code to real-world scenarios drastically.

Another reason is the idea of reusability. Code reusability ensures that a clean code is provided to the programmer.

This also helps in the reduction of rewriting and serves a bug-free code, as the replication of the code gets reduced to almost zero with the help of reusability. Other advantages that can be achieved through reusability are time management, maintenance, and ease of extension. You can do manipulations and add some desired features to a class that already exists through inheritance. One more reason is the transitive nature of inheritance. Transitive nature implies that if two objects that are in succession show a pattern, then all the objects of that order must show the exact pattern.

For example, if a new class Tata Safari has been declared as a subclass of Car which itself is a subclass of Vehicle, then Tata Safari must also be a Vehicle i.e., inheritance is transitive in nature. Access modifiers and encapsulation are concepts in object-oriented programming (OOP) that help control the visibility and accessibility of properties and methods in a class. These concepts are used to enforce encapsulation, which is one of the pillars of OOP.

Access Modifiers:

Access modifiers define the visibility of properties and methods in a class. In PHP, there are three main access modifiers:

1. Public (public):

- A public property or method can be accessed from outside the class.

- Example:

```
php    class MyClass {  
public $publicProperty;  
  
    public function publicMethod() {  
        // code here  
    }  
}
```

2. Protected (protected):

- A protected property or method can only be accessed within the class itself or within its subclasses (inheritance).

- Example:

```
php class MyClass {    protected
$protectedProperty;

    protected function protectedMethod() {
        // code here
    }
}
```

3. Private (private):

- A private property or method can only be accessed within the class itself.

- Example:

```
php class MyClass {
private $privateProperty;

    private function privateMethod() {
        // code here
    }
}
```

Encapsulation:

Encapsulation is the bundling of data (properties) and the methods that operate on the data into a single unit, known as a class. The idea is to hide the internal implementation details of the class and expose only what is necessary. Access modifiers play a crucial role in achieving encapsulation.

Example of Encapsulation in PHP:

```
php class Car {    private
$model;    private
$color;

    public function setModel($model) {        // Perform
validation or other logic if needed
        $this->model = $model;
    }

    public function getModel() {        return
$this->model;
    }

    public function setColor($color) {        // Perform
validation or other logic if needed
        $this->color = $color;
    }

    public function getColor() {        return
$this->color;
    }
}

// Usage
$myCar = new Car();
$myCar->setModel("Toyota");
$myCar->setColor("Blue");

echo "Model: " . $myCar->getModel(); // Output: Model: Toyota echo
"Color: " . $myCar->getColor(); // Output: Color: Blue
```

In this example, the properties \$model and \$color are private, and their values can only be accessed and modified through the public methods setModel, getModel, setColor, and getColor. This ensures that the internal state of the Car class is controlled and validated according to the class's logic.

✓ **Abstraction**

An abstract class is one of the most exciting and important topics of the OOPs. A class is said to be an abstract class if it contains at least one abstract method. Abstract methods are the methods that do not contain a body. Abstract classes totally rely on the derived class to carry out their tasks.

An abstract class is created when you only have the method name but you are not certain how to write the code for the same.

The following example will illustrate the working of the abstract classes in PHP.

```
<?php

// define an abstract class

abstract class Furniture {

    public $name;

    public function __construct($name) {

        $this->name = $name;

    }

    // abstract method of the

    // abstract class.

    // It will be defined later in the

    // child classes.

    abstract public function printType() : string;

}
```

```

// Derived classes are defined now

class Sofa extends Furniture {

    public function printType() : string {

        return "I am a $this->name!";

    }

}

class Table extends Furniture {

    public function printType() : string {

        return "I am a $this->name!";

    }

}

class Cupboard extends Furniture {

    public function printType() : string {

        return "I am a $this->name!";

    }

}

// Creating instances of the

// derived classes.

$Sofa = new Sofa("Sofa");

echo $Sofa->printType();

echo PHP_EOL;

```

```
$Table = new Table("Table");

echo $Table->printType();


echo PHP_EOL;

$Cupboard = new Cupboard("Cupboard");

echo $Cupboard->printType();

echo PHP_EOL;

?>
```



```
I am a Sofa!
I am a Table!
I am a Cupboard!

...Program finished with exit code 0
Press ENTER to exit console.█
```

Polymorphism

Learning Outcome 2: Connect PHP to the Database

IC 2.1: Application of Database Connection drives

MySQLi

MySQLi (MySQL Improved) is a PHP extension used to communicate with MySQL databases. It offers both procedural and object-oriented interfaces. **Important Information**

1. MySQLi is specific to MySQL databases and offers both procedural and object-oriented approaches.
2. MySQLi - OOP is the object-oriented version of MySQLi, providing a more modern way to interact with databases using objects and methods.
3. PDO is a database abstraction layer that supports multiple database drivers, making it more flexible if you need to switch databases in the future.

Procedural Style:

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

// Create connection
$conn = mysqli_connect ($servername, $username, $password, $dbname);
// Check connection if (!$conn) { die ("Connection failed: "
. mysqli_connect_error());
} echo "Connected successfully";
?>
```

Object-Oriented Style:

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

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

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

MySQLi - OOP (Object-Oriented Programming)

Object-Oriented Programming (OOP) style provides a more intuitive and flexible way to work with databases. It uses objects and methods to interact with the database.

```

<? php
// Database connection parameters
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "database_name";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error);
}
// Example query
$sql = "SELECT * FROM table_name"; $result
= $conn->query($sql); if ($result->num_rows >
0) {
// Output data
while($row = $result->fetch_assoc()) {
echo "id: " . $row["id"]. " - Name: " . $row["name"]. "<br>";
}
}

else { echo "0 results";
}
$conn->close();
?>

```

PDO (PHP Data Objects)

PDO provides a database abstraction layer that allows for more database drivers than just MySQL. It supports multiple databases like MySQL, PostgreSQL, SQLite, and more.


```

<? php
// Database connection parameters
$servername = "localhost";
$username = "username";
$password = "password"; $dbname =
"database_name";
try {
// Create a PDO instance
$conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
// Set the PDO error mode to exception
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); echo
"Connected successfully";
// Example query
$sql = "SELECT * FROM table_name";
$stmt = $conn->query($sql);
// Fetch results
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
echo "id: " . $row['id'] . " - Name: " . $row['name'] . "<br>";
}
}
catch(PDOException $e) { echo "Connection failed: "
. $e->getMessage();
}
// Close connection
$conn = null;
?>

```

IC 2.2: Perform database CRUD Operations

CRUD stands for Create, Read, Update, and Delete, which are the four basic operations that can be performed on data in most relational database systems. In PHP, you can interact with a database to perform these CRUD operations using various methods and libraries.

CRUD with MySQLi (Procedural)

Create

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

$conn = mysqli_connect($servername, $username, $password, $dbname); if
(!$conn) { die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO users (username, email) VALUES ('JohnDoe',
'john.doe@example.com')"; if
(mysqli_query($conn, $sql)) { echo "New
record created successfully";
} else { echo "Error: " . $sql . "<br>" .
mysqli_error($conn);
} mysqli_close($conn); ?>
```

Read

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

$conn = mysqli_connect($servername, $username, $password, $dbname); if
(!$conn) { die("Connection failed: " . mysqli_connect_error());
}
}
```

```

$sql = "SELECT id, username, 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["username"]. " - Email: " .
$row["email"]. "<br>";
    }
} else {      echo "0
results";
} mysqli_close($conn);
?>

```

Update

```

<? php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = mysqli_connect($servername, $username, $password, $dbname); if
(!$conn) {      die("Connection failed: " . mysqli_connect_error());
}

$sql = "UPDATE users SET email='john.doe.new@example.com' WHERE
username='JohnDoe'"; if (mysqli_query($conn, $sql)) {      echo "Record updated
successfully";
} else {      echo "Error updating record: " .
mysqli_error($conn);
} mysqli_close($conn);
?>

```

Delete

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

$conn = mysqli_connect($servername, $username, $password, $dbname); if
(!$conn) { die("Connection failed: " . mysqli_connect_error());
}

$sql = "DELETE FROM users WHERE username='JohnDoe'"; if
(mysqli_query($conn, $sql)) { echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

CRUD with MySQLi (OOP)

Create, Read, Update, Delete (CRUD) operations using Object-Oriented

Programming (OOP) with MySQLi are similar to the procedural approach but use the \$conn-> syntax.

Create

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

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
    $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

```

$sql = "INSERT INTO users (username, email) VALUES ('JohnDoe',
'john.doe@example.com')"; $conn->exec($sql);
echo "New record created successfully";
} catch(PDOException $e) {    echo "Error:
" . $e->getMessage();
}
$conn = null;
?>

```

Read, Update, Delete

```

<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
    $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Read
    $stmt = $conn->prepare("SELECT id, username, email FROM users");
        $stmt->execute();    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "id: " . $row["id"]. " - Name: " . $row["username"]. " - Email: " .
    $row["email"]. "<br>";
        }
// Update
    $sql = "UPDATE users SET email='john.doe.new@example.com' WHERE
    username='JohnDoe'; $conn->exec($sql);    echo "Record updated successfully";

```

```

// Delete
$sql = "DELETE FROM users WHERE username='JohnDoe'";
$conn->exec($sql);    echo "Record
deleted successfully"; }
catch(PDOException $e) {    echo "Error: " .
    $e->getMessage();
}
$conn = null;
?>

```

Sure! Below is an example of how you can import and export a MySQL database using procedural PHP with MySQLi.

Export Database

```

<? php
$host = 'localhost';
$user = 'username';
$password = 'password';
$dbase = 'database_name';
// Create connection
$conn = mysqli_connect($host, $user, $password, $dbase);
// Check connection if (!$conn) {    die("Connection failed: "
    . mysqli_connect_error());
}
// Backup filename
$backup_file = 'database_backup_' . date("Y-m-d_H-i-s") . '.sql';
// Execute MySQL dump exec("mysqldump --user={$user} --
password={$password} --host={$host}
{$dbase} > {$backup_file}"); echo "Database exported
successfully to {$backup_file}"; mysqli_close($conn); ?>

```

Import Database

```
<? php
$host = 'localhost';
$user = 'username';
$password = 'password';
$database = 'database_name';
// Create connection
$conn = mysqli_connect($host, $user, $password, $database);

// Check connection if (!$conn) { die("Connection failed: "
. mysqli_connect_error());
}
// SQL file to import
$sql_file = 'path_to_sql_file.sql';
// Read SQL file
$sql_contents = file_get_contents($sql_file);
// Execute multi-query if (mysqli_multi_query($conn,
$sql_contents)) { do {
    // Store and free result set if ($result =
mysqli_store_result($conn)) {
mysqli_free_result($result);
}
} while (mysqli_next_result($conn)); echo
"Database imported successfully.";
} else { echo "Error importing database: " .
mysqli_error($conn);
} mysqli_close($conn);
?>
```

Notes for direction

1. Replace 'username', 'password', and 'database_name' with your MySQL username, password, and database name respectively.
2. For exporting, the code uses mysqldump command-line utility to create a backup of the database.
3. For importing, the code reads the SQL file and executes the queries using mysqli_multi_query().
4. Make sure you have appropriate permissions to execute mysqldump and access the SQL file.

IC 2.3: Application of PHP Basic security concepts

1. Input Validation

Input validation is crucial to prevent malicious data from being processed by your PHP application. Always validate and sanitize user input before using it in your application.

Example:

```
<? php
$username = $_POST['username']; if (preg_match('/^[a-zA-Z0-9]+$/', $username)) {
    // Valid username
} else {
    // Invalid username
}
```

2. Password Security

Storing passwords securely is essential to protect user data. Always hash passwords using strong cryptographic algorithms like bcrypt or Argon2.

Example:

```
<? php
$password = $_POST['password'];
$hashed_password = password_hash($password, PASSWORD_BCRYPT);
```


3. Cross-Site Scripting (XSS) Prevention

Prevent XSS attacks by escaping output data that could be manipulated to execute malicious scripts.

Example: `<? php echo htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8'); ?>`

4. Cross-Site Request Forgery (CSRF) Prevention

Prevent CSRF attacks by generating and validating unique tokens for each form submission.

Example: `<? php
session_start();
$token = bin2hex(random_bytes(32));
$_SESSION['csrf_token'] = $token;
// In the form
<input type="hidden" name="csrf_token" value="<?php echo $token;
?>`

5. Session Security

Ensure session data is secure by setting appropriate session configurations and using HTTPS to encrypt data in transit.

Example: `<? php
session_set_cookie_params([
 'lifetime' => 86400,
 'path' => '/',
 'domain' => '.example.com',
 'secure' => true,
 'httponly' => true,
 'samesite' => 'Strict'
]); session_start();
?>`

6. File Uploads

Secure file uploads by restricting file types, validating file size, and storing files outside the web root directory.

Example:

```
<? php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check file size if ($_FILES["fileToUpload"]["size"] >
500000) {    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
// Allow certain file formats if($imageFileType != "jpg" && $imageFileType != "png"
&& $imageFileType !=
"jpeg"
&& $imageFileType != "gif" ) {    echo "Sorry, only JPG, JPEG, PNG
& GIF files are allowed.";
    $uploadOk = 0;
}
```

7. Error Reporting

Limit the information exposed to users by disabling detailed error messages in production environments.

Example: <? php

```
ini_set('display_errors', 'Off'); error_reporting(0);
?>
```

IC 2.4: Errors and exceptions in PHP

Introduction

Errors and exceptions are a common part of programming, including PHP. They occur when something unexpected happens during the execution of a script. Proper error handling is essential for debugging and maintaining the quality of your PHP applications.

Types of Errors

1. **Syntax Errors:** These occur when the PHP parser encounters a syntax mistake in the code. They prevent the script from running.
2. **Runtime Errors:** Also known as exceptions, these occur during the execution of the script. They can be due to issues like division by zero, accessing undefined variables, etc.
3. **Logical Errors:** These are the hardest to detect as they don't cause PHP to raise errors. Instead, they lead to incorrect output or behavior of the program due to flawed logic in the code.

Exception Handling

Exception handling in PHP allows you to handle runtime errors gracefully by catching and managing exceptions rather than letting them terminate the script abruptly. Try-Catch Blocks:

```
<? php try {  
    // Code that may throw an exception  
} catch (Exception $e) {    // Handle the  
exception    echo "Error: " . $e-  
>getMessage();  
} ?>
```

Simple “die ()” statements

The die () function is a simple way to halt the execution of a script and display a message. It's often used for basic error handling or to stop the script when a certain condition is met.

```
<? php  
$file = 'somefile.txt'; if  
(!file_exists($file)) {  
die("File not found");  
}
```

Custom Error and Error Triggers

You can define custom error handlers in PHP using `set_error_handler()` function. This allows you to handle errors in a way that suits your application.

```
<? php function customErrorHandler($errno, $errstr) {  
    echo "Error: [$errno] $errstr";  
}  
  
// Set custom error handler set_error_handler("customErrorHandler");  
// Trigger an error  
trigger_error("This is a custom error", E_USER_ERROR);
```

Error Reporting

PHP provides `error_reporting()` function to set the level of error reporting during script execution. You can use it to control which errors are displayed or logged.

```
<? php  
// Report all errors error_reporting(E_ALL);  
  
// Report all errors except notices error_reporting(E_ALL &  
~E_NOTICE);
```

IC 2.5: Implementation of user authentication

Implementation of User Authentication

Introduction

User authentication is the process of verifying the identity of a user who is trying to access a system, application, or network. It ensures that only authorized users can access the resources and services, protecting them from unauthorized access.

Types of User Authentication

1. Password-based Authentication: Users authenticate by providing a username and password.
2. Multi-factor Authentication (MFA): Requires users to provide two or more verification factors (e.g., password, SMS code, fingerprint).
3. Biometric Authentication: Uses unique biological traits like fingerprints, face recognition, or iris scans.

4. Token-based Authentication: Users authenticate using a token, often used in OAuth or JWT (JSON Web Token).
5. Certificate-based Authentication: Uses digital certificates issued by a Certificate Authority (CA) to authenticate users.

User Authorization

User authorization determines what actions or resources a user is allowed to access after successful authentication. It defines permissions and roles for different users, ensuring that users can only access the information and perform actions that they are authorized to.

Create User Authentication

Start a Session

In a web application, a session is typically started when a user logs in. A session ID is generated and stored on the server, while a cookie containing this session ID is sent to the user's browser. <? php session_start();

?>

Authenticate the User

To authenticate the user, you need to verify the provided credentials against the stored credentials in the database.

<? php

// Assuming \$username and \$password are obtained from the login form

\$username = \$_POST['username'];

\$password = \$_POST['password'];

// Retrieve user details from the database

// Check if the username exists //

Verify the password if

(\$authenticated) {

 \$_SESSION['loggedin'] = true;

 \$_SESSION['username'] = \$username;

```

// Redirect to the protected page
header('Location: protected_page.php');
} else {
    // Invalid credentials, redirect to login page with error message    header('Location:
login.php?error=invalid_credentials');
}
?>

```

Protect Pages

To protect pages and ensure only authenticated users can access them, you can check the session variable at the beginning of each protected page.

```

<? php session_start();
if (!isset($_SESSION['loggedin']) || $_SESSION['loggedin'] !== true) {
    // User is not logged in, redirect to login page
    header('Location: login.php');    exit;
}

```