ADVANCED INFORMATION TECHNOLOGY INSTITUTE

GHANA-INDIA KOFI ANNAN CENTRE OF EXCELLENCE IN ICT

# *INTRODUCTION TO PHP*

# Table of Contents


1. **Introduction**
    1. What is PHP?
    2. Why PHP?
2. **PHP – Environment Setup**
    1. Xampp installation
3. **Variables, Operators, Constants and Expressions.**
    1. Syntax
    2. Data types
    3. Operators
    4. Variables

## 4. Conditional Statements

1. If Statement
2. If Else Statement
3. Elseif Statement
4. Switch Statement

### Loops

1. While Loop
2. Do while Loop
3. For Loop
4. Foreach Loop
5. Break/Continue Loop

## 5. Arrays

6. **Functions**

7. **PHP Forms**
    1. Form Handling
    2. Form Validation
    3. Form Required

8. **PHP GET and POST Methods**

9. **Connection to MySQL Database**

10. **PHP Object-Oriented Programming(OOP)**

# WHAT IS PHP?

- **PHP** is a server-side scripting language designed for web development but also used as a general-purpose programming language. Originally created by Rasmus Lerdorf in 1994, the PHP reference implementation is now produced by The PHP Group. PHP originally stood for Personal Home Page, but it now stands for the recursive backronym PHP: **Hypertext Preprocessor**. PHP code may be embedded into HTML code, or it can be used in combination with various web template systems, web content management systems and web frameworks. PHP code is usually processed by a PHP interpreter implemented as a module in the web server or as a Common Gateway Interface (CGI) executable.
- It allows web developers to create dynamic content that interacts with databases. **PHP** is basically used for developing web based software applications. **PHP** is mainly focused on server-side scripting, so you can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. Code is executed in servers, that is why you'll have to install a sever-like environment enabled by programs like XAMPP which is an Apache distribution.

# WHY PHP?

1. **Compatible with almost all servers used nowadays**
   A web server is an information technology that processes requests via HTTP, the basic network protocol used to distribute information on the World Wide Web. There exist many types of web servers that servers use. Some of the most important and well-known are: Apache HTTP Server, IIS (Internet Information Services), lighttpd, Sun Java System Web Server etc. As a matter of fact, PHP is compatible with all these web servers and many more.
2. **PHP supports a wide range of databases**
   An important reason why PHP is so used today is also related to the various databases it supports (is compatible with). Some of these databases are: DB++, dBase, Ingres, Mongo, MaxDB, MongoDB, mSQL, MySQL, OCI8, PostgreSQL, SQLite, SQLite3, Oracle, Microsoft SQL Server and so on.
3. **PHP is free to download and open source**
   PHP is an open source project – the language is developed by a worldwide team of volunteers who make its source code freely available on the Web, and it may be used without payment of licensing fees or investments in expensive hardware or software. This reduces software development costs without affecting either flexibility or reliability. The open-source nature of the code further means that any developer, anywhere , can inspect the code tree, spit errors, and suggest possible fixes, this produces a stable, robust product wherein bugs, once discovered, are rapidly resolved – sometimes within a few hours of discovery.

## 4. PHP will run on most platforms:

Unlike some technologies that require a specific operating system or are built specifically for that, PHP is engineered to run on various platforms like Windows, Mac OSX, Linux, Unix etc). As a result, a PHP application developed on, say, Windows will typically run on UNIX without any significant issues. This ability to easily undertake cross-platform development is a valuable one, especially when operating in a multi platform corporate environment or when trying to address multiple market segments.

## 5. Performance:

Scripts written in PHP executives faster than those written in other scripting language, with numerous independent benchmarks, putting the language ahead of competing alternatives like JSP, ASP.NET and PERL. The PHP 5.0 engine was completely redesigned with an optimized memory manager to improve performance, and is noticeable faster than previous versions.

## 6. Easy to learn & large community:

PHP is a simple language to learn step by step. This makes it easier for people to get engaged in exploring it. Its syntax is clear and consistent, and it comes with exhaustive documentation for the 5000+ functions included with the core distributions. This significantly reduces the learning curve for both novice and experienced programmers, and it's one of the reasons that PHP is favored as a rapid prototyping tool for Web-based applications It also has such a huge community online that is constantly willing to help you whenever you're stuck (which actually happens quite a lot).

# XAMPP INSTALLATION SETUP

**XAMPP** is a free and open source cross-platform web server solution developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages. In order to make your PHP code execute locally, first install XAMPP.

### XAMPP Server Installation Steps.

- Download the XAMPP Setup at https://www.apachefriends.org/download.html
- Double click the xampp-windows-x64-8.1.5-0-VS16-installer.exe file.
- Choose a language from the menu and click Ok.
- Click on Next button.
- Click on Next button again
- Click on install and finally click the Finish button.
- Click Yes, to open the XAMPP Control Panel.
- Now click on start for Apache and MySQL.
- To Launch, open your browser and type http://localhost/. you will get to the welcome page.

# *PHP* ENVIRONMENT *SETUP*

**How to execute PHP Script on XAMPP Server.**

- **Step 1 :** First Create a PHP file called ( index.php ) using any text editor like VS Code, notepad, notepadd++, Sublime text.
- **Step 2 :** Write a PHP script inside index.php file. Example;

```
<?php
        echo   "Welcome to the new world of PHP";
?>
```

- **Step 3 :** How to save index.php file. Create a folder called myproject inside C:xampp/htdocs/myproject
  - **Save the index.php file inside:** C:xampp/htdocs/myproject/index.php

- **Step 4 :** Open your browser and type in the url: **localhost/myproject/index.php**

# PHP SYNTAX

Syntax is a way to representation of PHP script. Basically it gives the primary idea to specify the code format. It also specify the area of a written code. A PHP scripts starts with <?php and ends with ?>. Example;

```
<?php

        // PHP code goes here

?>
```

A PHP file normally contains HTML tags, and some PHP scripting code. Below, we have an example of a simple PHP file, with a PHP script that uses a built-in function called "echo" to output the text "Hello World" on a web page.

```
<!DOCTYPE html>
    <html>
        <body>

            <h1>My first PHP page</h1>

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

        </body>
    </html>
```

# DATA TYPES

Data types specify the size and type of values that can be stored. To check only data type use **gettype( )** function. The PHP **var_dump()** function returns the data type and value.

## PHP DATA TYPES

- String
- Integer
- Float / Double
- Boolean
- Array
- Object
- Null
- Resource

**STRING:-** A string is a sequence of characters, like "Hello world!". A string can be any text inside quotes. You can use single or double quotes.

```
<?php
    $name  = "John Doe ";
    echo $name;
?>
```

**INTEGER:-** An integer data type is a non-decimal number. Integer means **numeric** data types. The size of an integer is platform-dependent, although a maximum value of about two billion is the usual value. The PHP var_dump() function returns the data type and value.

```
<?php
        $num=  100;
        var_dump($num);
?>
```

**FLOAT/DOUBLE:-** A float (floating point number) is a number with a decimal point or a number in exponential form. In the following example **$num** is a float. The PHP var_dump() function returns the data type and value.

```
<?php
    $num  =  10.67;
    var_dump($num);
?>
```

**BOOLEAN:-** A Boolean represents two possible states: TRUE or FALSE.

```
<?php
    $old =  True;
    $new = Flase;
?>
```

**ARRAY:-** An array stores multiple values in one single variable. In the following example, **$numbers** and **$cars** are arrays. The PHP var_dump() function returns the data type and value.

```
<?php
    $numbers = array(10,20,30,40,50);

    $cars = array("Volvo","BMW","Toyota");

    var_dump($numbers);
    var_dump($cars);

?>
```

**OBJECT:-** Classes and objects are the two main aspects of object-oriented programming. A class is a template for objects, and an object is an instance of a class. When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties. Let's assume we have a class named Car. A Car can have properties like model, color, etc. We can define variables like $model, $color to hold the values of these properties.

## OBJECT

```php
<?php
class Car {
    public $color;
    public $model;
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
    }
    public function message() {
        return "My car is a " . $this->color . " " . $this->model . "!";
    }
}

$myCar = new Car("black", "Volvo");
echo $myCar -> message();
echo "<br>";
$myCar = new Car("red", "Toyota");
echo $myCar -> message();
?>
```

**NULL:-** Null is a special data type which can have only one value: NULL. A variable of data type NULL is a variable that has no value assigned to it. If a variable is created without a value, it is automatically assigned a value of NULL. Variables can also be emptied by setting the value to NULL.

```
<?php
        $blank = null;
        var_dump($blank);
?>
```

**RESOURCE:-** The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common example of using the resource data type is a database call.

```
<?php
        $connection = mysqli_connect("localhost", "root", "password", "schooldatabase");
?>
```

# OPERATORS

**Operators** are symbols that tell the PHP processor to perform certain actions. Operators are used to perform operations on variables and values. For example, the **addition(+)** symbol is an **Operators** that tells PHP to **add** two variables or values, **while** the **greater-than(>)** symbol is an **Operators** that tells PHP to **compare** two values.

## TYPES OF OPERATORS

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Increment/Decrement operators
- Array operators

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

| Operators | Name |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus |
| ** | Exponentiation |

**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.

| Operators | Name |
|---|---|
| += | Add and assign |
| -= | Subtract and assign |
| *= | Multiply and assign |
| /= | Divide and assign quotient |
| % | Divide and assign modulus |
| .= | Concatenate and assign(its used only for string) |

## Add and Assign

```
<?php
        $num = 500;
        $num += 500;
    echo  "sum="  .$num.  "</br>";
?>
```

## Subtract and Assign

```
<?php
        $num = 1000;
        $num  -=500;
    echo  "Subtraction = "  .$num.  "</br>";
?>
```

**COMPARISON OPERATORS:-** The PHP comparison operators are used to compare two values (number or string).

| Operators | Name | Example | Result |
|---|---|---|---|
| == | Equal | $x == $y | Returns true if $x is equal to $y |
| === | Identical | $x === $y | Returns true if $x is equal to $y, and they are of the same type |
| != | Not equal | $x != $y | Returns true if $x is not equal to $y |
| <> | Not equal | $x <> $y | Returns true if $x is not equal to $y |
| !== | Not identical | $x !== $y | Returns true if $x is not equal to $y, or they are not of the same type |
| > | Greater than | $x > $y | Returns true if $x is greater than $y |
| < | Less than | $x < $y | Returns true if $x is less than $y |
| >= | Greater than or equal to | $x >= $y | Returns true if $x is greater than or equal to $y |
| <= | Less than or equal to | $x <= $y | Returns true if $x is less than or equal to $y |
| <=> | Spaceship | $x <=> $y | Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. |

**LOGICAL OPERATORS:-** The PHP logical operators are used to combine conditional statements.

| Operators | Name | Example | Result |
|---|---|---|---|
| and | And | $x and $y | True if both $x and $y are true |
| or | Or | $x or $y | True if either $x or $y is true |
| xor | Xor | $x xor $y | True if either $x or $y is true, but not both |
| && | And | $x && $y | True if both $x and $y are true |
| \|\| | Or | $x \|\| $y | True if either $x or $y is true |
| ! | Not | !$x | True if $x is not true |

**INCREMENT / DECREMENT OPERATORS:-** The PHP increment operators are used to increment a variable's value and the decrement operators are used to decrement a variable's value.

| Operators | Name | Description |
|---|---|---|
| ++$x | Pre-increment | Increments $x by one, then returns $x |
| $x++ | Post-increment | Returns $x, then increments $x by one |
| --$x | Pre-decrement | Decrements $x by one, then returns $x |
| $x-- | Post-decrement | Returns $x, then decrements $x by one |

**ARRAY OPERATORS:-** The PHP array operators are used to compare arrays.

| Operators | Name | Example | Result |
|---|---|---|---|
| + | Union | $x + $y | Union of $x and $y |
| == | Equality | $x == $y | Returns true if $x and $y have the same key/value pairs |
| === | Identity | $x === $y | Returns true if $x and $y have the same key/value pairs in the same order and of the same types |
| != | Inequality | $x != $y | Returns true if $x is not equal to $y |
| <> | Inequality | $x <> $y | Returns true if $x is not equal to $y |
| !== | Non-identity | $x !== $y | Returns true if $x is not identical to $y |

# PHP VARIABLES

Variables are "containers" used to store both numeric and non-numeric information;

**Rules for Variable declaration**

1. Variables in PHP starts with a **dollar($)** sign, followed by the name of the variable.
2. The variable name must begin with a letter or the underscore character.
3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ ).
4. A variable name should not contain space.
5. Variable names are case-sensitive ( $age and $AGE are two different variables ).

A variable is created the moment you assign a value to it: Example

```
<?php
        $text  =  “Hello World”;
        $num  =  500;
        $dec   =  10.5;
?>
```

# PHP VARIABLES 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:**

1. Local
2. Global
3. Static

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

```
<?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>";
?>
```

# PHP VARIABLES SCOPE

**Static Scope:** 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();


?>
```

**Exercise**

Create a variable named txt and assign the value “Hello World”

# CONDITIONAL STATEMENTS

Conditional statements are used to perform different actions based on different conditions.

**In PHP, we have the following conditional statements:**

1. **The if Statement** : In if Statements Output will appear when only Condition must be true.
2. **The if-else Statement** : if-else statements allows you to display output in both the condition(if condition is true display some message otherwise display other message).
3. **The if-elseif-else Statement** : The if-else-if-else statement lets you chain together multiple if-else statements, thus allowing the programmer to define actions for more than just two possible outcomes.
4. **The switch Statement** : The switch statement is similar to a series of if statements on the same expression.

**IF STATEMENT:-** The **if** statement executes the code if one condition is true. Example:

```
<?php
    $t = date("H");

    if ($t < "20") {

        echo "Have a good day!";

    }
?>
```

**IF-ELSE STATEMNT:-** The **if-else** statement executes some code if a condition is true and another code if that condition is false. Example:

```
<?php

        $t = date("H");

        if ($t < "20") {

            echo "Have a good day!";

        }else    {
            echo  “Have a good night!”;
            }
?>
```

**IF---ELSEIF----ESLE STATEMENT:-** The **if----elseif---else** statement executes different codes for more than two conditions.

```
<?php
    $t = date("H");
    if ($t < "10") {
        echo "Good morning!";
    }elseif($t < "20")  {
        echo "Have a good day!";
    }else  {
        echo "Have a good night";
?>
```

**Exercise-:** Fill in the blank spaces.

```
$x = 50;
$y = 10
        ----- ------ > ----- {
            echo "Hello World";
        }
```

**THE SWITCH STATEMENT:-** The **switch** statement is used to perform different actions based on different conditions. Example;

```
<?php
        $color = "red";

        switch ($color) {
                case "red":
                        echo "Your favorite color is red!";
                        break;

                 case "blue":
                        echo "Your favorite color is blue!";
                        break;

                case "green":
                        echo "Your favorite color is green!";
                        break;

                default:
                        echo "Your favorite color is neither red, blue, nor green!";
        }
?>
```

# LOOPS

Loops are used to execute the same block of code again and again, as long as a certain condition is true. When you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task.

**These are the six types of loop used in PHP:**

1. **While Loop-:** while loop executes the statement while a specified condition is true. In while loop first check the condition then execute the statement. Example:

```
<?php
        $num  =  0;
        while($num  <=  100)  {
                echo  "The number is : $num <br>";
                $num+= 10;
?>
```

**Example explained:**

- $num = 0;  Initialize the loop counter ($num), and set the start value to 0
- $num  <= 100;  Continue the loop as long as $num is less than or equal to 100
- $num+ = 10;  Increase the loop counter value by 10 for each iteration.

2. **Do-While-:** do - while loop executes the statement once and next statement depends upon a specified condition is true. In do - while loop first execute the statement then check the condition, it means if condition is false in that case one time statement execute. Example:

```
<?php
        $num   =   1;
        do  {
                echo  "The number is : $num <br>";
                 $num++;
            }
         while ( $num  <=  5 );
?>
```

3. **For Loop-:** loops through a block of code a specified number of times. Example;

```
<?php
        for ( $num = 0;   $num  <=  10;   $num++)  {
                echo   "The number is :  $num  <br>";
        }
?>
```

**Example explained:**

- $num = 0; Initialize the loop counter ($x), and set the start value to 0.
- $num  <= 10;  Continue the loop as long as $num is less than or equal to 10
- $num++ ;  Increase the loop counter value by 1 for each iteration.

4. **Foreach Loop-:** Loops through a block of code for each element in an array. The foreach loop works only on arrays, and is used to loop through each key/value pair in an array. Example;

```
<?php
        $colors = array("red", "green", "blue", "yellow");

        foreach ($colors as $value) {
                echo "$value <br>";
        }
?>
```

5. **Break Loop-:** The **break** can be use to jump out of a loop. This example jumps out of the loop when **$num** is equal to **4**:

```
<?php
        for ( $num = 0; $num < 10; $num++ ) {

                if ( $num == 4 ) {
                        break;
                }

                echo "The number is : $num <br> ";
        }
?>
```

6. **Continue Loop-:** The **continue** statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop. This example skips the value of **4** :

```
<?php
    for ( $num = 0; $num < 10; $num++ ) {

        if ( $num == 4 ) {
            continue;
        }

        echo "The number is : $num <br> ";
    }
?>
```

# ARRAYS

An array is a special variable, which can hold more than one value at a time. The **PHP Array Functions** is basically an inbuilt function in the PHP language that is used in order to create an array.

**There three types of Arrays in PHP:**

1. **Indexed arrays** - Arrays with a numeric index.
2. **Associative arrays** - Arrays with named keys.
3. **Multidimensional arrays** - Arrays containing one or more arrays.

1. **Indexed Arrays:** There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this

```
<?php
    $cars = array("Volvo", "BMW", "Toyota");
?>
```

or the index can be assigned manually:

```
<?php
    $cars[0] = "Volvo";
    $cars[1] = "BMW";
    $cars[2] = "Toyota"
?>
```

Example: To loop through and print all the values of an indexed array, you could use a for loop like this;

```
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    $arrlength = count($cars);

    for($x = 0; $x < $arrlength; $x++) {
        echo $cars[$x];
        echo "<br>";
    }
?>
```

2. **Associative Arrays:** Associative arrays are arrays that use named keys that you assign to them. There are two ways to create an associative array.

```
<?php
        $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
 ?>
```

or:

```
<?php
        $age['Peter'] = "35";
        $age['Ben'] = "37";
        $age['Joe'] = "43"
 ?>
```

Example: To loop through and print all the values of an associative array, you could use a **forecach** loop like this;

```
<?php
        $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

            foreach($age as $x => $x_value) {
                    echo "Key=" . $x . ", Value=" . $x_value;
                     echo "<br>";
            }

 ?>
```

3. **Multidimensional Arrays:** A multidimensional array is an array containing one or more arrays.
Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column . To get access to the elements of the $cars array we must point to the two indices (row and column).

```
<?php

        echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
        echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
        echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
        echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";

?>
```

Example: We can also put a **for loop** inside another **for loop** to get the elements of the $cars arrays(we still have to point to the two indices).

```
<?php

    for ($row = 0; $row < 4; $row++) {
                echo "<p><b>Row number $row</b></p>";
                echo "<ul>";
        for ($col = 0; $col < 3; $col++) {
                echo "<li>".$cars[$row][$col]."</li>";
         }
                echo "</ul>";
    }

 ?>
```

# FUNCTIONS

**Functions** are self contained block of statement which used to perform any specific task.

**Types of Functions:**

1. System defined/library/inbuilt
2. User defined

**Advantages Of Function**

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

# Create a User Defined Function in PHP

A user-define function declaration starts with the word **function**.
**Note:** A function name must start with a letter or an underscore. Function names are NOT case-sensitive. Example;

In the example below, we create a function named "writeMsg()". The opening curly brace ( { ) indicates the beginning of the function code, and the closing curly brace ( } ) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name followed by brackets ().

```
<?php
        function writeMsg() {
                echo "Hello world!";
        }
                writeMsg(); // call the function
?>
```

## PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma. The following example has a function with two arguments ($fname, $year). When the familyName() function is called, we also pass along a name and a year (e.g. Jani, 1990). The name and the year are used inside the function, which outputs several different first names.

```
<?php
        function familyName($fname,  $year) {
                echo "$fname  was born in  $year   <br>";
        }

                familyName("Jani“,   “1990”);
                familyName("Hege“,   “2010”);
?>
```

# PHP FORM HANDLING

**PHP form** is used to take input from users. in PHP if you want to take input from keyboard and display the output according to input, in that case you have to use html form. html form's have a property : form method in which you have to set either get or post method.

**The example below displays a simple HTML form with two input fields and a submit button:**

```
<html>
    <body>

        <form action="welcome.php" method="post">
            Name: <input type="text" name="name"><br>
            E-mail: <input type="text" name="email"><br>
                <input type="submit">
        </form>

    </body>
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

# PHP FORM REQUIRED AND VALIDATION

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers.

**The example below displays a simple HTML form with four input fields and a submit button:**

```
<html>
    <body>
        <form action="welcome.php" method="post">

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

            E-mail: <input type="text" name="email“  required ><br>

            Comment: <textarea name="comment" rows="5" cols="40“  required></textarea><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
                <input type="submit">

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

# PHP GET AND POST METHODS

**GET METHOD:** Information sent from a form with the **GET** method is **visible to everyone** (all variable names and values are displayed in the URL). **GET** also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
**GET** may be used for sending non-sensitive data. Example;

```
<html>
    <body>

        <form action="welcome.php" method="GET">
            Name: <input type="text" name="name"><br>
            E-mail: <input type="text" name="email"><br>
                <input type="submit">
        </form>

    </body>
</html>
```

**Note:** **GET** should NEVER be used for sending passwords or other sensitive information!
Using GET method limited data sends. **GET** method is faster way to send data.

# PHP GET AND POST METHODS

**POST METHOD:** **POST** method is secured method because it hides all information. Using POST method unlimited data sends. **POST** method is slower method compare to GET method. Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send. Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

**POST** may be used for sending sensitive data. Example;

```
<html>
    <body>
        <form action="welcome.php" method="POST">

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

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

            Comment: <textarea name="comment" rows="5" cols="40"  required></textarea><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
                <input type="submit">

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

# PHP MYSQL DATABASE

With PHP, you can connect to and manipulate databases. **MySQL** is the most popular database system used with PHP.

### What is MySQL?

- **MySQL** is a database system used on the web.
- **MySQL** is a database system that runs on a server.
- **MySQL** is ideal for both small and large applications.
- **MySQL** is very fast, reliable, and easy to use.
- **MySQL** uses standard SQL.
- **MySQL** compiles on a number of platforms.
- **MySQL** is free to download and use.
- **MySQL** is developed, distributed, and supported by Oracle Corporation.

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.
MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia). Another great thing about MySQL is that it can be scaled down to support embedded database applications.

# CONNECTION TO MYSQL DATABASE

**Open a Connection to MySQLi Object-Oriented.**

Before we can access data in the MySQL database, we need to be able to connect to the server.
Example **(MySQLi Object-Oriented).**

```
<?php
        $servername = "localhost";
        $username = "username";
        $password = "password";
        $myDB =  “school”;

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

        // Check connection
        if (($conn->connect_error) {

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

        }
        echo "Connected successfully";

?>
```

# CONNECTION TO MYSQL DATABASE

**Open a Connection to MySQLi Procedural**

Before we can access data in the MySQL database, we need to be able to connect to the server.
Example **(MySQLi Procedural).**

```
<?php
        $servername = "localhost";
        $username = "username";
        $password = "password";
        $myDB=  “school”;

        // Create connection
        $conn = mysqli_connect($servername, $username, $password, $myDB);

        // Check connection
        if (!$conn) {

        die("Connection failed: " .    mysqli_connect_error() );

        }
        echo "Connected successfully";

?>
```

# CONNECTION TO MYSQL DATABASE

**Open a Connection to PDO( PHP DATA OBJECTS)**

Before we can access data in the MySQL database, we need to be able to connect to the server.

Example **PDO ( PHP DATA OBJECTS ).**

```
<?php
        $servername = "localhost";
        $username = "username";
        $password = "password";
        $myDB =  “school”;
    try   {
        // Create connection
        $conn = new  PDO(“mysql:host=$servername;dbname=myDB; $username; $password”);
                // set the PDO error mode to exception

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


                    echo "Connected successful”;
            // Check connection
        }      catch (PDOException $e) {

                echo "Connected failed :  “  .  $e->getMessage();
            }

?>
```

# PHP Object-Oriented Programming(OOP)

**Object-oriented programming** is a computer programming model that organizes software design around data, or objects.

### Advantages of Object-Oriented Programming(OOP)

1. OOP is faster and easier to execute.
2. OOP provides a clear structure for the programs.
3. OOP is convenient because security is built-in with encapsulation. Other methods and classes cannot access private data by default, and programs written in OOP languages are more secure for it.
4. Hiding of information.
5. OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug.
6. OOP makes it possible to create full reusable applications with less code and shorter development time.

**Note**: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

# Principles of Object-Oriented Programming(OOP)

**Object-oriented programming is based on the following principle:**

- **Encapsulation:** This principle states that all important information is contained inside an object and only select information is exposed. The implementation and state of each object are privately held inside a defined class. Other objects do not have access to this class or the authority to make changes. They are only able to call a list of public functions or methods. This characteristic of data hiding provides greater program security and avoids unintended data corruption.
- **Abstraction:** Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code. The derived class can have its functionality extended. This concept can help developers more easily make additional changes or additions over time.
- **Inheritance:** Classes can reuse code from other classes. Relationships and subclasses between objects can be assigned, enabling developers to reuse common logic while still maintaining a unique hierarchy. This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.
- **Polymorphism:** Objects are designed to share behaviors and they can take on more than one form. The program will determine which meaning or usage is necessary for each execution of that object from a parent class, reducing the need to duplicate code. A child class is then created, which extends the functionality of the parent class. Polymorphism allows different types of objects to pass through the same interface.

**Examples of object-oriented programming languages?**

- Java, C++, C#, Ruby,
- Python, PHP, TypeScript etc.

# Benefits of Object-Oriented Programming(OOP)

**Benefits of Object-oriented programming include:**

- **Modularity:** Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.
- **Reusability:** Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.
- **Productivity:** Programmers can construct new programs quicker through the use of multiple libraries and reusable code.
- **Easily upgradable and scalable:** Programmers can implement system functionalities independently.
- **Interface descriptions:** Descriptions of external systems are simple, due to message passing techniques that are used for objects communication.
- **Security:** Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and internet protocols are protected.
- **Flexibility:** Polymorphism enables a single function to adapt to the class it is placed in. Different objects can also pass through the same interface.

Classes and objects are the two main aspects of object-oriented programming.

# WHAT ARE CLASSES AND OBJECTS(OOP)

**A Class** is a template or a blueprint of an object and an **Object** is an instance of a class. Example;

| Class | Objects |
|---|---|
| Fruit | Apple<br>Banana<br>Mango |

| Class | Objects |
|---|---|
| Car | Toyota<br>Benz<br>Hyundai |

# CLASS IN PHP

Below we declare a **class** named **Fruit** consisting of two properties ($name and $color) and two methods **set_name**() and **get_name**() for setting and getting the $name property.

```php
<?php
    class Fruit {
        // Properties
        public $name;
        public $color;

        // Methods
        function set_name($name) {
            $this->name = $name;
        }
        function get_name() {
            return $this->name;
        }
    }
?>
```

**Note:** **Variables** within a **class** are called properties and **functions** are called methods.

# OBJECTS IN PHP

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values. Example;

```php
<?php
class Fruit {
    // Properties
    public $name;
    public $color;

    // Methods
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
}
    $apple = new Fruit();
    $banana = new Fruit();
    $apple->set_name('Apple');
    $banana->set_name('Banana');

    echo $apple->get_name();
    echo "<br>";
    echo $banana->get_name();
?>
```

## Example of a Class and an Object in PHP

```php
<?php
class Fruit {
    // Properties
    public $name;
    public $color;

    // Methods
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
    function set_color($color) {
        $this->color = $color;
    }
    function get_color() {
        return $this->color;
    }
}

    $apple = new Fruit();
    $apple->set_name('Apple');
    $banana->set_color('Red');

    echo "Name: " . $apple->get_name();
    echo "<br>";
    echo "Color: " . $apple>get_color();
?>
```

# CONSTRUCTOR IN PHP(OOP)

A constructor allows you to initialize an object's properties upon creation of the object. If you create a __construct() function, PHP will automatically call this function when you create an object from a class.

**Note:** that the construct function starts with two underscores (__). Example;

```php
<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name,  $color) {
        $this->name = $name;
        $this->color  = $color;

    function get_name() {
        return $this->name
    }
    function get_color() {
        return $this->color;
    }
}
    $apple = new Fruit("Apple",  "Red");
    echo $apple->get_name();
    echo "<br>";
    echo $apple->get_color();
?>
```

# DESTRUCTOR IN PHP(OOP)

A destructor is called when the object is destructed or the script is stopped or exited. If you create a __destruct() function, PHP will automatically call this function at the end of the script.

**Note:** that the destruct function starts with two underscores (__).

The example below has a __construct() function that is automatically called when you create an object from a class, and a __destruct() function that is automatically called at the end of the script.

```
<?php
        class Fruit {
                public $name;
                public $color;

                function __construct($name,  $color) {
                        $this->name = $name;
                        $this->color  = $color

                function __destruct() {
                        echo   "The fruit is  {$this->name} and the color is  {$this->color}. ";
                }

        }


                $apple = new Fruit("Apple",  "Red");

?>
```

# ACCESS MODIFIERS IN PHP(OOP)

**Access Modifier** allows you to alter the visibility of any class member(properties and method). Properties and methods can have access modifiers which control where they can be accessed.

**There are three access modifiers:**

1. **Public:** Public access modifier is open to use and access inside the class definition as well as outside the class definition. Moreover, the property or method can be accessed from everywhere. This is default.
2. **Protected:** Protected is only accessible within the class in which it is defined and its parent or inherited classes.
3. **Private:** Private is only accessible within the class that defines it.( it can't be access outside the class means in inherited class).

```php
<?php
class Fruit {
    public $name;
    public $color;
    private $weight
        public function set_name($n) {          // a public function (default)
            $this->name = $n;
        }
        protected function set_color($n) {      // a protected function
            $this->color = $n;
        }
        private function set_weight($n) {       // a private function
            $this->weight = $n;
        }
}
$mango = new Fruit();
$mango->set_name('Mango');  // Ok
?>
```

# INHERITANCE IN PHP(OOP)

**Inheritance** is a mechanism of extending an existing class by inheriting a class we create a new class with all functionality of that existing class, and we can add new members to the new class. An inherited class is defined by using the **extends** keyword. Example;

```
<?php
        class Fruit {
                public $name;
                public $color;
                public function __construct($name,  $color) {
                        $this->name = $name;
                        $this->color  = $color
                }
                public function  intro() {
                        echo   "The fruit is  {$this->name} and the color is  {$this->color}. ";
                }
        }

        class Strawberry extends Fruit  {
                public function message()   {
                        echo   "Am I a fruit or a strawberry";
                }
        }
                $strawberry =  new Strawberry("Strawberry",   "Red");
                $strawberry->message();
                $strawberry->intro();
?>
```

# ABSTRACT CLASSES IN PHP(OOP)

**Abstraction** is a way of hiding information. An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code. Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.

**Note:** An Abstract class or method is define with the **abstract** keyword.

```
<?php
        class Car{
                public $name;
                public function __construct($name) {
                        $this->name = $name;
                }
                abstract public function intro() : string;
        }

        class Toyota extends Car{
                public function intro() : string  {
                        return  "Choose Ghana product!  I'm a  $this->name!";
                }
        }



                // Create object from the child classes
                $toyota  =  new Toyota("Toyoto Thundra");
                echo  $Toyota->intro();


                $strawberry->intro();
?>
```

# INTERFACES IN PHP(OOP)

**An Interface** is a mechanism of extending an existing class by inheriting a class we create a new class with all functionality of that existing class, and we can add new members to the new class. Interfaces allow you to specify what methods a class should implement. Interfaces make it easy to use a variety of different classes in the same way. When one or more classes use the same interface, it is referred to as "polymorphism".

**Note:** An Interface is declare with the **interface** keyword.

```php
<?php
interface Animal{
        public function makeSound() {

        }
}
class Dog implements Animal{
        public function makeSound()  {
                echo " Bark ";
        }
}
class Cat implements Animal{
        public function makeSound()  {
                echo " Meow ";
        }
}
        $dog = new Dog();
        $cat = new Cat();
$animals = array($dog, $cat);

foreach($animals as $animal) {
        $animal->makeSound();
}
?>
```