

Information and Communication Technology

Grade 13

Competency 10.7: PHP and MySQL

Contents

Introduction

Install PHP

Basic Syntax

- Insert PHP code into the HTML
- Comments in PHP

Identifiers

Variables

- Variables Scope
 - Local
 - Global
 - Static

Data Types

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array

Operators

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

Control Structures

- Conditional Statements
 - if statement
 - if...else statement

- if...elseif....else statement
 - switch statement
- Control Loops
 - While
 - do...while
 - for
 - foreach

Arrays

- Indexed arrays
 - Loop Through an Indexed Array
- Associative arrays
 - Loop Through an Associative Array

Functions

- Create a User Defined Function
- Function Arguments
- Default Argument Value
- Functions - Returning values

Database connectivity

Introduction to PHP

PHP stands for Hypertext Preprocessor (no, the acronym doesn't follow the name). It is an open source, server-side, scripting language used for the development of web applications. PHP is an HTML-embedded Web scripting language. This means PHP code can be inserted into the HTML of a Web page. PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.) It is compatible with almost all servers used today (Apache, IIS, etc.) and supports a wide range of databases. Download it from the official PHP resource: www.php.net



Introduction to PHP File

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

Install PHP

- install a web server, such as apache
- install PHP
- install a database, such as MySQL

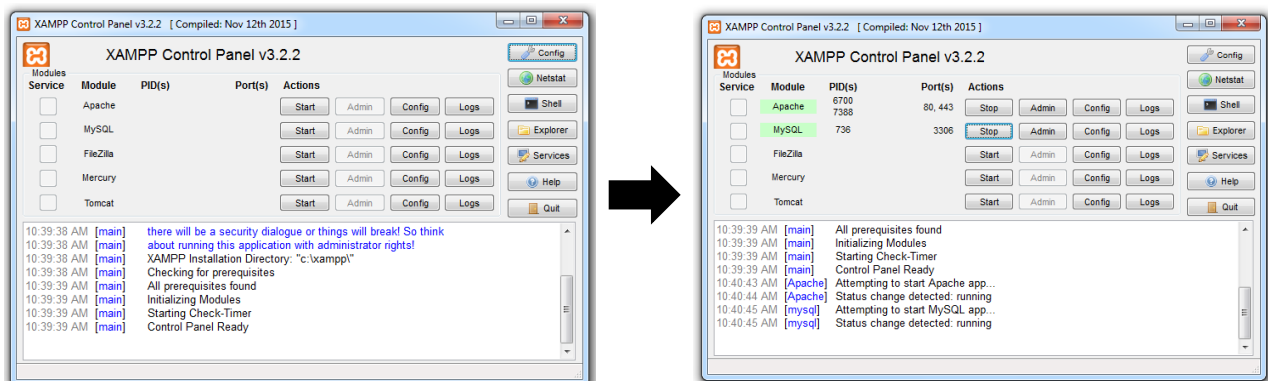
Otherwise install XAMPP or WAMP server to your computer. Web server, PHP and database are contained in XAMPP and WAMP. So you do not need to configure web server, PHP and database separately. Web server, PHP and database are automatically configured by XAMPP and WAMP servers.

<u>XAMPP</u> 	 <u>WAMP</u>
XAMPP stands for x-os, apache, mysql, php , perl. (x-os means it can be used for any operating system.)	WAMP. Stands for "Windows, Apache, MySQL, and PHP.
https://www.apachefriends.org/index.html	http://www.wampserver.com/en/

➤ Install XAMPP



➤ Start Apache and MySQL servers



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  
?>
```
- PHP statements end with a semicolon (;)
- All keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive but all variable names are case-sensitive.

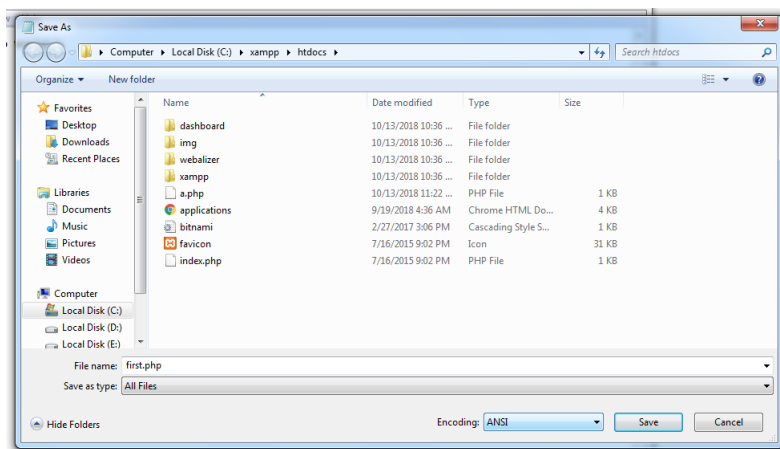
Write Simple PHP code and show the result

This script uses a built-in PHP function "echo" to output the text "Hello ICT Students!" on a web page.

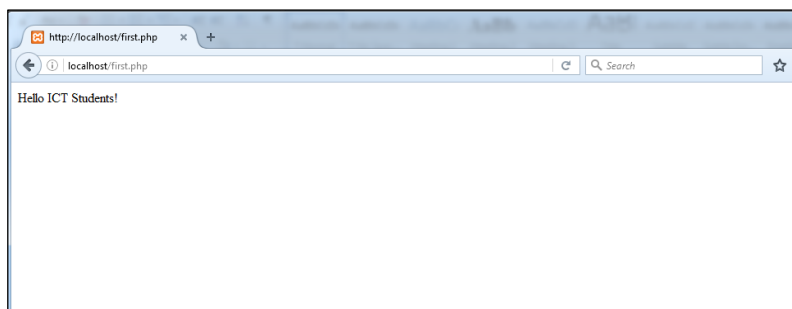
- 1) First open notepad and type below mentioned PHP code

```
<?php  
    echo "Hello ICT Students!";  
?>
```

- 2) Save that script into the **xampp\htdocs** folder (Normally it is located in C:\xampp\htdocs). Enter the file name as **first.php** being sure to include the .php extension. Set the Save As Type to **All Files**. Finally, click the Save button.



- 3) After that open the web browser (Google Chrome, Mozilla Firefox, etc) and type <http://localhost/first.php> in the address bar and press enter key



Insert PHP code into the HTML

```

<html>

  <head>

    <title>PHP</title>

  </head>

  <body>

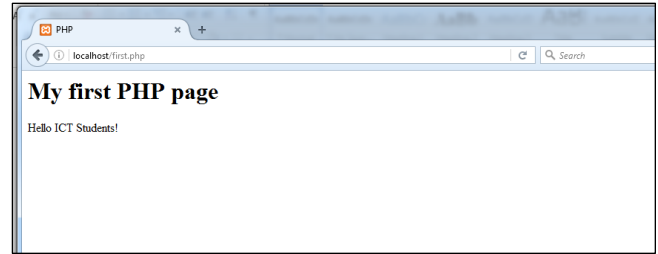
    <h1>My first PHP page</h1>

    <?php
      echo "Hello ICT Students!";
    ?>

  </body>

</html>

```



Comments in PHP

```

<html>

  <body>

    <?php

      // This is a single-line comment

      # This is also a single-line comment

      /*
      This is a multiple-lines comment block
      that spans over multiple
      lines
      */

      // You can also use comments to leave out parts of a code line
      $x = 5 /* + 15 */ + 5;
      echo $x;

    ?>

  </body>

</html>

```

Output

10

Creating (Declaring) PHP Identifiers

An identifier is a term used to represent a variable or a program. The following are some rules that should be followed in Declaring an Identifier.

- A identifier must start with a letter or the underscore character
- A identifier cannot start with a number
- A identifier can only contain alpha-numeric characters and underscores (A-Z, a-z, 0-9, and _)
- The special characters such as the following should not be included in an identifier.
` ~ ! @ # \$ % ^ & * - () + = [] { } : ; ' " < > ? . , / | \
- There should not be any space between words
- An identifiers are case-sensitive (\$age, \$Age and \$AGE are three different identifiers)
- Use of meaningful terms for identifiers can make program easily understood

Examples of valid identifiers

\$sum, \$Grade, \$TOTAL, \$Main_counter, \$_discount

Examples of invalid identifiers

\$Gross Amount, \$9Address, \$Paid-Amount

PHP Variables

A variable is an identifier which changes the values given to it, when the program is being executed. A variable starts with the \$ sign, followed by the name of the variable.

Ex :-

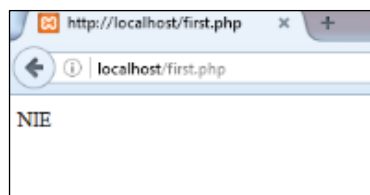
```
$discount = 10;  
$subject_name = "Information and Communication Technology";  
$average = 45.86;
```

The PHP `echo` statement is often used to output data to the screen.

`<?php`

```
$name="NIE";  
echo $name;
```

`?>`



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:

- local
- global
- static

Local variables

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

Example 01:

`<?php`

```
function LocalTest() {  
    $x = 5; // local scope  
}  
LocalTest ();  
  
echo $x; // using x outside the function will generate an error
```

`?>`

This script will generate an error. Because \$x is a local variable and it cannot be accessed from outside the function.

Example 02:

`<?php`

```
function LocalTest () {  
    $x = 5; // local scope  
    echo $x;  
}  
LocalTest ();
```

`?>`

Output 5

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

Global variables

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

Example 01:

```
<?php
```

```
$a = 10; $b = 15; /* global scope */
```

```
function GlobalTest(){
```

```
    echo $a; /* reference to local scope variable */
```

```
}
```

```
GlobalTest ();
```

```
?>
```

Notice : Undefined variable : a in C:\xampp\htdocs\global.php on line 6

This script will not produce any output because the echo statement refers to a local version of the `$a` variable, and it has not been assigned a value within this scope. In PHP the `global` keyword is used to access a global variable from within a function.

Example 01 Using *global*

```
<?php
```

```
$a = 10; $b = 15;
```

```
function GlobalTest (){
```

```
    global $a, $b;
```

```
    echo $a;
```

```
}
```

```
GlobalTest ();
```

```
?>
```

The above script will output **10**. By declaring `$a` global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function.

Static variables

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 StaticTest() {
```

```
static $x = 0;
echo $x;
$x++;
}
```

```
StaticTest ();
StaticTest ();
StaticTest ();
?>
```

PHP Data Types

When a program is executed, the input and output should be stored in computer memory. The space needed for each is defined according to the data type. Variables can store data of different types, and different data types can do different things. PHP supports the following data types

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array

String

A string is a sequence of characters. A string can be any text that means string can hold letters, numbers, and special characters. The way to specify a string is to enclose it in single quotes (e.g. 'Hello world!') or double quotes ("Hello world!").

Example:

```
<?php
```

```
$x = "Hello world!";
$y = 'Hello world!';
```

```
echo $x;
echo "<br>";
echo $y;
```

```
?>
```

Output

Hello world!

Hello world!

Integer

Integers are whole numbers, without a decimal point (between -2,147,483,648 and 2,147,483,647). Integers can be specified in decimal (base 10), hexadecimal (base 16), octal (base 8) or binary (base 2) notation. An integer can be either positive or negative (... , -2, -1, 0, 1, 2, ...).

Example:

The PHP `var_dump()` function returns the data type and value

```
<?php
```

```
$a = 1234; // decimal number
var_dump ($a);
echo "<br>";

$b = -123; // a negative number
var_dump ($b);
echo "<br>";

$c = 0123; // octal number (equivalent to 83 decimal)
var_dump ($c);
echo "<br>";

$d = 0x1A; // hexadecimal number (equivalent to 26 decimal)
var_dump ($d);
echo "<br>";

$e = 0b11111111; // binary number (equivalent to 255 decimal)
var_dump ($e);
echo "<br>";
```

Output

```
int(1234)
int(-123)
int(83)
int(26)
int(255)
```

```
?>
```

Floating Point Numbers or Doubles

Floating point numbers (also known as "floats", "doubles") are decimal or fractional numbers.

Example:

```
<?php
```

```
$x = 10.365;
var_dump($x);
echo "<br>";
```

Output

```
float(10.365)
float(10200)
```

```
$y = 10.2e3;  
var_dump($y);
```

?>

Boolean

A Boolean represents two possible states: TRUE or FALSE.

Example:

```
<?php  
$x = true;  
var_dump($x);  
echo "<br>";  
  
$y = false;  
var_dump($y);
```

?>

Output

```
bool(true)  
bool(false)
```

Array

An array stores multiple values in one single variable. An array is formally defined as an indexed collection of data values. Each index (also known as the key) of an array is unique and references a corresponding value.

The PHP `print_r()` function prints human-readable information about a variable

Example :

```
<?php  
$cars = array("Volvo","BMW","Toyota");  
print_r($cars);  
echo "<br>";  
  
$color_codes = array (  
    "Red" => "#ff0000",  
    "Green" => "#00ff00",  
    "Blue" => "#0000ff" );  
print_r($color_codes);
```

?>

Output

```
Array ( [0] => Volvo [1] => BMW [2] => Toyota )  
Array ( [Red] => #ff0000 [Green] => #00ff00 [Blue] => #0000ff )
```

Manipulating PHP Strings

PHP provides many built-in functions for manipulating strings like calculating the length of a string, reversing a string, replacing part of a string with different characters and many others.

- **Calculating the Length of a String**

The *strlen()* function is used to calculate the number of characters inside a string. It also includes the blank spaces inside the string.

Example:

```
<?php
```

```
$my_str = 'Welcome to PHP';
```

```
echo strlen($my_str);
```

```
?>
```

Output

14

- **Counting Number of Words in a String**

The *str_word_count()* function counts the number of words in a string.

Example:

```
<?php
```

```
echo str_word_count("Hello world!");
```

```
?>
```

Output

2

- **Reversing a String**

The *strrev()* function reverses a string.

Example:

```
<?php
```

```
echo strrev("Hello world!");
```

```
?>
```

Output

!dlrow olleH

- **Replace Text Within a String**

The *str_replace()* replaces all occurrences of the search text within the target string.

Example:

The example below replaces the text " Students" with "Sri Lanka":

```
<?php
```

```
$my_str = "Hello Students!";  
echo str_replace("Students", "Sri Lanka", $my_str);
```

```
?>
```

Output

Hello Sri Lanka!

PHP Constants

Constants are like variables except that once they are defined they cannot be changed or undefined. Constants are very useful for storing data that doesn't change while the script is running. Constants are defined using PHP's `define()` function, which accepts three arguments: the name of the constant, its value and case-insensitivity. Once defined the constant value can be accessed at any time just by referring to its name. Unlike variables, constants are automatically global across the entire script.

`define(name, value, case-insensitive)`

Parameters:

- name: Specifies the name of the constant
- value: Specifies the value of the constant
- case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false

Example 01 : Creates a constant with a **case-sensitive** name:

```
<?php
```

```
define("GREETING", "Welcome to PHP!");  
echo GREETING;
```

```
?>
```

Output

Welcome to PHP!

Example 02 : Creates a constant with a **case-insensitive** name:

```
<?php
```

```
define("GREETING", "Welcome to PHP!", true);  
echo greeting;
```

Output

Welcome to PHP!

?>

PHP Operators

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

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

Arithmetic Operators

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

<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Description</i>	<i>Result</i>
+	Addition	5+6	Sum of 5 and 6	11
-	Subtraction	8-6	Difference of 8 and 6	2
*	Multiplication	5*2	Product of 5 and 2	10
/	Division	10/4	Quotient of 10 and 4	2.5
%	Modulus	15%4	Remainder of 15 divided by 4	3
**	Exponentiation	2**5	Result of raising 2 to the 5'th power	32

Example :

<?php

```
$x = 5;  
$y= 2;
```

```
echo $x+$y;  
echo $x-$y;  
echo $x*$y;  
echo $x/$y;  
echo $x%$y;
```

Output

7
3
10
2.5
1
25

```
echo $x**$y;
```

```
?>
```

Assignment operators

The assignment operators are used to assign values to variables. 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.

| <i>Operator</i> | <i>Name</i> | <i>Example</i> | <i>Same as ...</i> |
|-----------------|---------------------------|----------------|--------------------|
| = | Assign | $\$x = \y | $\$x = \y |
| += | Add and assign | $\$x += \y | $\$x = \$x + \$y$ |
| -= | Subtract and assign | $\$x -= \y | $\$x = \$x - \$y$ |
| *= | Multiply and assign | $\$x *= \y | $\$x = \$x * \$y$ |
| /= | Divide and assign | $\$x /= \y | $\$x = \$x / \$y$ |
| %= | Divide and assign modulus | $\$x \% = \y | $\$x = \$x \% \$y$ |

Example :

```
<?php
```

```
$x = 10;  
$y = 5;
```

```
echo $x;    // Outputs: 10
```

```
$x += $y;  
echo $x;    // Outputs: 15
```

```
$x = 8;  
$y = 5;  
$x -= $y;  
echo $x;    // Outputs: 3
```

```
$x = 8;  
$y = 3;  
$x *= $y;  
echo $x;    // Outputs: 24
```

```
$x = 17;  
$y = 5;  
$x /= $y;  
echo $x;    // Outputs: 3.4
```



```
$x = 8;
$x %= 3;
echo $x;    // Outputs: 2
```

?>

Comparison operators

The PHP comparison operators are used to compare two values (number or string). The final result always takes a Boolean value. So result will be True or False.

<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Result</i>
==	Equal	<code>\$x == \$y</code>	Returns true if \$x is equal to \$y
===	Identical	<code>\$x === \$y</code>	Returns true if \$x is equal to \$y, and they are of the same type
!=	Not equal	<code>\$x != \$y</code>	Returns true if \$x is not equal to \$y
<>	Not equal	<code>\$x != \$y</code>	Returns true if \$x is not equal to \$y
!==	Not identical	<code>\$x !== \$y</code>	Returns true if \$x is not equal to \$y, or they are not of the same type
>	Greater than	<code>\$x > \$y</code>	Returns true if \$x is greater than \$y
<	Less than	<code>\$x < \$y</code>	Returns true if \$x is less than \$y
>=	Greater than or equal to	<code>\$x >= \$y</code>	Returns true if \$x is greater than or equal to \$y
<=	Less than or equal to	<code>\$x <= \$y</code>	Returns true if \$x is less than or equal to \$y

Example :

<?php

```
$x = 25;
$y = 35;
$z = "25";

var_dump($x == $z);    // Outputs: boolean true
var_dump($x === $z);   // Outputs: boolean false
var_dump($x != $y);    // Outputs: boolean true
var_dump($x !== $z);   // Outputs: boolean true
var_dump($x < $y);     // Outputs: boolean true
var_dump($x > $y);     // Outputs: boolean false
var_dump($x <= $y);    // Outputs: boolean true
var_dump($x >= $y);    // Outputs: boolean false
```

?>

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.

<i>Operator</i>	<i>Name</i>	<i>Description</i>
<code>++\$x</code>	Pre-increment	Increments \$x by one, then returns \$x
<code>\$x++</code>	Post-increment	Returns \$x, then increments \$x by one
<code>--\$x</code>	Pre-decrement	Decrements \$x by one, then returns \$x
<code>\$x--</code>	Post-decrement	Returns \$x, then decrements \$x by one

Example :

<?php

```
$x = 10;
echo ++$x;    // Outputs: 11
echo $x;      // Outputs: 11

$x = 10;
echo $x++;    // Outputs: 10
echo $x;      // Outputs: 11

$x = 10;
echo --$x;    // Outputs: 9
echo $x;      // Outputs: 9

$x = 10;
echo $x--;    // Outputs: 10
echo $x;      // Outputs: 9
```

?>

Logical operators

The logical operators are typically used to combine conditional statements.

<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Result</i>
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

Example :

Control Structures

A control structure is a block of programming that analyzes variables and chooses a direction in which to go based on given parameters. There are two types of Control Structures in PHP: Conditional Statements and Control Loops.

1. Conditional Statements

Conditional Statements allow to branch the path of execution in a script based on whether a single, or multiple conditions, evaluate to true or false. The following are the basic Conditional Statements in the PHP.

- if statement
- if...else statement
- if...elseif...else statement
- switch statement

if statement

The if statement executes some code if one condition is true

Syntax

```
if (condition) {  
    code to be executed if condition is true;  
}
```

Example :

The example below will output "You are pass the exam" if the marks is greater than fifty(50).

```
<?php  
$marks = 78;  
  
if ($marks > 50) {  
    echo " You are pass the exam";  
}
```

Output

You are pass the exam

```
}  
?>
```

if...else statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
If (condition) {  
    code to be executed if condition is true;  
} else {  
    code to be executed if condition is false;  
}
```

Example :

The example below will output "You are pass the exam" if the marks is greater than fifty (50) and "Sorry! you are fail the exam" otherwise.

```
<?php  
$marks = 45;  
  
if ($marks > 50) {  
    echo "You are pass the exam";  
} else {  
    echo "Sorry! you are fail the exam";  
}  
?>
```

Output

Sorry! You are fail the exam

if...elseif....else statement

The if....elseif...else statement executes different codes for more than two conditions.

Syntax

```
if (condition) {  
    code to be executed if this condition is true;  
} elseif (condition) {
```

```
    code to be executed if this condition is true;
} else {
    code to be executed if all conditions are false;
}
```

Example :

The example below will output "A" if the marks is greater than or equal to 75, and "B" if the marks is greater than or equal to 65, and "C" if the marks is greater than or equal to 50, and "S" if the marks is greater than or equal to 35. Otherwise it will output "W".

```
<?php
$marks = 49;

if ($marks >= 75) {
    echo "A";
}elseif ($marks >= 65) {
    echo "B";
} elseif ($marks >= 50) {
    echo "C";
} elseif ($marks >= 35) {
    echo "S";
} else{
    echo "W";
}
?>
```

Output

S

Switch statement

The `switch` statement is used to perform different actions based on different conditions. The switch-case statement is an alternative to the if-elseif-else statement, which does almost the same thing. The switch-case statement tests a variable against a series of values until it finds a match, and then executes the block of code corresponding to that match.

This is how it works: First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use `break` to prevent the code from running into the next case automatically. The `default` statement is used if no match is found.

Syntax

```
switch (n) {  
    case label1:  
        code to be executed if n=label1;  
        break;  
    case label2:  
        code to be executed if n=label2;  
        break;  
    case label3:  
        code to be executed if n=label3;  
        break;  
    ...  
    default:  
        code to be executed if n is different from all labels;  
}  
}
```

Example :

```
<?php  
$today = "Tue";  
  
switch ($today) {  
    case "Mon":  
        echo "Today is Monday. Wake up early in the morning.";  
        break;  
    case "Tue":  
        echo "Today is Tuesday. Do your homework.";  
        break;  
    case "Wed":  
        echo "Today is Wednesday. Play a game";  
        break;  
    case "Thu":  
        echo "Today is Thursday. Its movie time";  
        break;  
}
```

```

case "Fri":
    echo "Today is Friday. Do some rest.";
    break;
default:
    echo "No information available for that day.";
}
?>

```

Output

Today is Tuesday. Do your homework.

2. Control Loops

PHP while loops execute a block of code while the specified condition is true. The following are the basic looping Statements in the PHP.

- While
- do...while
- for
- foreach

While Loop

Conditions are checked at the beginning of the loop. Statements inside the loop are executed only if the condition is true. Statements inside the loop never executed if the condition is false. If the condition does not become false while the loop is executed, it will be an infinite loop.

Syntax

```

while (condition is true) {
    code to be executed;
}

```

Example 01:

The example below first sets a variable \$x to 0 (\$x = 0). Then, the while loop will continue to run as long as \$x is less than, or equal to 5 (\$x <= 5). \$x will increase by 1 each time the loop runs (\$x++):

```

<?php
$x = 0;

while($x <= 5) {
    echo "The number is: $x <br>";
}

```

Output

The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

```
$x++;  
}  
?>
```

Example 02:

```
<?php  
$x = 10;  
  
while($x >= 50) {  
    echo "The number is: $x <br>";  
    $x++;  
}  
?>
```

loop never executed

Example 03:

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

infinite loop

do...while Loop

The **do...while** loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

Syntax

```
do {  
    code to be executed;  
} while (condition is true);
```

Example 01:

The example below first sets a variable \$x to 2 (\$x = 2). Then, the do while loop will write some output, and then increment the variable \$x with 2. Then the condition is checked (is \$x less than, or equal to 10?), and the loop will continue to run as long as \$x is less than, or equal to 10. It will be displayed even numbers between 2 to 10

```
<?php  
$x = 2;
```

Output

```
2  
4  
6  
8  
10
```



```
do {
    echo " $x <br>";
    $x +=2;
} while ($x <= 10);
?>
```

Example 02:

```
<?php
$x = 6;
```

Output
6

```
do {
    echo " $x";
    $x++;
} while ($x <= 5);
?>
```

for loop

PHP for loops execute a block of code a specified number of times. The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
    code to be executed;
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

Example 01:

The example below displays the numbers from 1 to 10

```
<?php
for ($x = 1; $x <= 10; $x++) {
    echo $x. " , ";
}
?>
```

Output

1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10

Example 02:

The example below displays the odd numbers from 1 to 10

```
<?php
for ($x = 1; $x <= 10; $x += 2) {
    echo $x. " , ";
}
?>
```

Output 1 , 3 , 5 , 7 , 9

foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array. For every loop iteration, the value of the current array element is assigned to \$value and the array pointer is moved by one, until it reaches the last array element.

Syntax

```
foreach ($array as $value) {
    code to be executed;
}
```

Example:

```
<?php
$foods = array("Bread", "Chocolate", "Biscuit", "Ice-Cream");

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

Output Bread Chocolate Biscuit Ice-Cream

Arrays

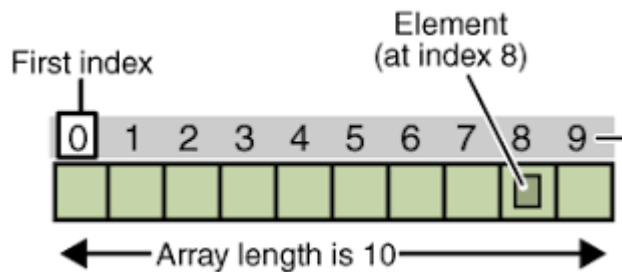
An array is a special variable, which can hold more than one value at a time. In PHP, the array() function is used to create an array.

In PHP, there are two types of arrays:

- Indexed arrays - Arrays with a numeric index
- Associative arrays - Arrays with named keys

- Multidimensional arrays - Arrays containing one or more arrays

Indexed Arrays



There are two ways to create indexed arrays. The index can be assigned automatically (index always starts at 0) or the index can be assigned manually

The index can be assigned automatically

```
$foods = array("Bread", "Chocolate", "Ice-Cream");
```

Index can be assigned manually

```
$foods [0] = "Bread";  
$foods [1] = "Chocolate";  
$foods [2] = "Ice-Cream";
```

Example 01:

```
<?php  
$foods = array("Bread", "Chocolate", "Ice-Cream");  
  
/*Display array elements by using indexes*/  
echo "I like " . $foods [0] . ", " . $foods [1] . " and " . $foods [2] . ".";  
?>
```

Output

I like Bread, Chocolate and Ice-Cream.

Example 02:

```
<?php  
$foods = array("Bread", "Chocolate", "Ice-Cream");  
  
/*Change array elements by using indexes*/
```

Output

I like Toffee, Chocolate and Cake.

```
$foods [0]= "Toffee";
$foods [2]= "Cake";
echo "I like " . $foods [0] . ", " . $foods [1] . " and " . $foods [2] . ".";
?>
```

- Get The Length of an Array - The count() Function

The count() function is used to return the length (the number of elements) of an array:

Example:

```
<?php
$foods = array("Bread", "Chocolate", "Ice-Cream");
echo count($foods);
?>
```

Output 3

- Loop Through an Indexed Array

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

Example:

```
<?php
$foods = array("Bread", "Chocolate", "Ice-Cream");
$arrlength = count($foods);

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

Output Bread Chocolate Ice-Cream
--

Associative arrays

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

```
$device = array("input"=>"Mouse", "output"=>"Monitor", "storage"=>"CD");
```

Or

```
$age['input'] = "Mouse";
$age['output'] = "Monitor";
$age['storage'] = "CD";
```

Example:

```
<?php
$device = array("input"=>"Mouse", "output"=>"Monitor", "storage"=>"CD");
echo $device['input'] . " is a input device.";
?>
```

Output

Mouse is a input device.

Loop Through an Associative Array

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

Example:

```
<?php
$device = array("input"=>"Mouse", "output"=>"Monitor", "storage"=>"CD");

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

Output

Key=input,
Value=Mouse
Key=output,
Value=Monitor

Functions

PHP has more than 1000 built-in functions. Besides the built-in PHP functions, we can create our own functions. A function is a block of statements that can be used repeatedly in a program. A function will not execute immediately when a page loads. A function will be executed by a call to the function.

- Create a User Defined Function
- Function Arguments
- Default Argument Value
- Functions - Returning values

Create a User Defined Function

A user-defined function declaration starts with the word function

Syntax

```
function functionName() {  
    code to be executed;  
}
```

A function name is an identifier. When creating a function name, have to follow identifier declaration rules.

Example:

Create a function named "writeMessage()". 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 ICT Students!". To call the function, just write its name.

```
<?php  
function writeMessage() {  
    echo "Hello ICT Students!";  
}  
  
writeMessage(); // call the function  
?>
```

Output Hello ICT Students!

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.

Syntax

```
function functionName($oneParameter, $anotherParameter) {  
    code to be executed;  
}
```

Example 01:

The following example has a function with one argument (\$fname). When the ShowName() function is called, we also pass along a name, and the name is used inside the function, which outputs several different first names.

```
<?php
function ShowName($fname) {
    echo "Your name is $fname <br>";
}
```

```
ShowName("Jack");
ShowName("Peter");
ShowName("Smith");
```

```
?>
```

Output

```
Your name is Jack
Your name is Peter
Your name is Smith
```

Example 02:

The following example has a function with two arguments (\$num1 and \$num2).

```
<?php
function getSum($num1 , $num2) {
    $sum = $num1 + $num2;
    echo "Sum of two numbers $num1 and $num2 is : $sum ";
}
```

```
getSum(25 , 56);
```

```
?>
```

Output

```
Sum of two numbers 25 and 56 is : 81
```

Default Argument Value

PHP allows to create functions with optional parameters — just insert the parameter name, followed by an equals (=) sign, followed by a default value.

Example:

```
<?php
function setAge($name , $age = 25) {
    echo "$name 's age is $age years old. <br>";
}
```

```
setAge("Peter" , 30);
setAge("Jack"); // will use the default value of 25
setAge("John" , 28);
```

```
?>
```

Output

```
Peter 's age is 30 years old.
Jack 's age is 25 years old.
John 's age is 28 years old.
```

Functions - Returning values

A function can return a value back to the script that called the function using the return statement. The value may be of any type, including arrays. To let a function return a value, use the *return* statement.

Example:

```
<?php
function getAverage($num1 , $num2, $num3) {
    $avg = ($num1 + $num2 + $num3)/3;
    return $avg;
}
```

Output

Average value is : 53.666666666667

```
echo "Average value is : " . getAverage(75, 30, 56);
?>
```

Database connectivity

PHP combined with MySQL are cross-platform. In order to store or access the data inside a MySQL database, you first need to connect to the MySQL database server. PHP offers two different ways to connect to MySQL server: **MySQLi** (Improved MySQL) and **PDO** (PHP Data Objects) extensions.

While the PDO extension is more portable and supports more than twelve different databases, MySQLi extension as the name suggests supports MySQL database only. MySQLi extension however provides an easier way to connect to, and execute queries on, a MySQL database server. Both PDO and MySQLi offer an object-oriented API, but MySQLi also offers a procedural API which is relatively easy for beginners to understand.

Syntax

```
$conn = mysqli_connect($servername, $username, $password, $database);
```

Example:

```
<?php
$servername = "localhost";
$username = "root";
$password = "1234";
$database= "school";

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

// Check connection
if ($conn === false) {
    die("Error : Connection failed: " . mysqli_connect_error());
}
```



```
}  
echo "Connected successfully";  
?>
```