|  |  |  |
| --- | --- | --- |
| C:\Users\Admin\Pictures\JATA KPM.png  **BAHAGIAN PENDIDIKAN TEKNIK DAN VOKASIONAL**  **KEMENTERIAN PENDIDIKAN MALAYSIA**  **ARAS 5 & 6, BLOK E14, KOMPLEKS E,**  **PUSAT PENTADBIRAN KERAJAAN PERSEKUTUAN**  **KERTAS PENERANGAN**  ***(INFORMATION SHEET)*** | | |
| **KOD DAN NAMA NOSS** | IT-010-3: 2016 APPLICATION MODULE DEVELOPMENT | |
| **KOD DAN NAMA CU / WA** | C02 - APPLICATION MODULE DEVELOPMENT | |
| **NAMA PROGRAM** | TEKNOLOGI SISTEM PENGURUSAN PANGKALAN DATA DAN APLIKASI WEB | |
| **TAHAP DAN SEMESTER** | 3 (SEMESTER 2) | |
| **KOD DAN TAJUK KURSUS** | KPD 2023 WEB PROGRAMMING | |
| **NO.D5N TAJUK KOMPETENSI** | K4 WRITE MODULE CODE | |
| **NO. KOD KSKV** | KPD2023 / P(8/13) | Muka Surat : 1 Drp : 13 |
| **NO. KOD NOSS** | IT-010-3:2016-C02/ P(8/13) |

**TAJUK/***TITLE***: WRITE MODULE CODE**

**TUJUAN/***PURPOSE***:**

Kertas penerangan ini adalah bertujuan menerangkan mengenai :

* Introduction to Basic Syntax
* PHP tags
* Echo
* Comments
* Define Variable
* Variable
* Constants
* Data types
* Variable Scope
* VariableVariables
* Operators
* Arithmetic Operator
* Assignment Operator
* Comparison Operator
* Logical Operator

**PENERANGAN/***INFORMATION***:**

**4.3 INTRODUCTION TO BASIC SYNTAX**

**PHP**

PHP is a server scripting language, and is a powerful tool for making dynamic and interactive Web pages quickly.

PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.**Syntax**

The PHP script is executed on the server, and the plain HTML result is sent back to the browser.

## 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  
?>

The default file extension for PHP files is ".php".

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 PHP function "echo" to output the text "Hello World!" on a web page:

## Example

<!DOCTYPE html>  
<html>  
<body>  
<h1>My first PHP page</h1>  
<?php  
echo "Hello World!";  
?>  
</body>  
</html>

**Note:** PHP statements are terminated by semicolon (;). The closing tag of a block of PHP code also automatically implies a semicolon (so you do not have to have a semicolon terminating the last line of a PHP block).

## Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is editing the code!

Comments are useful for:

* To let others understand what you are doing - Comments let other programmers understand what you were doing in each step (if you work in a group)
* To remind yourself what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports three ways of commenting:

## Example

<!DOCTYPE html>  
<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 more than  
one line  
\*/  
?>  
  
</body>  
</html>

**PHP Case Sensitivity**

In PHP, all user-defined functions, classes, and keywords (e.g. if, else, while, echo, etc.) are NOT case-sensitive.

In the example below, all three echo statements below are legal (and equal):

## Example

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
ECHO "Hello World!<br>";  
echo "Hello World!<br>";  
EcHo "Hello World!<br>";  
?>  
  
</body>  
</html>

However; in PHP, all variables are case-sensitive.

In the example below, only the first statement will display the value of the $color variable (this is because $color, $COLOR, and $coLOR are treated as three different variables):

## Example

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$color="red";  
echo "My car is " . $color . "<br>";  
echo "My house is " . $COLOR . "<br>";  
echo "My boat is " . $coLOR . "<br>";  
?>  
  
</body>  
</html>

# 4.4 DEFINE VARIABLES

Variables are "containers" for storing information:

## Example

<?php  
$x=5;  
$y=6;  
$z=$x+$y;  
echo $z;  
?>

## Much Like Algebra

x=5  
y=6  
z=x+y

In algebra we use letters (like x) to hold values (like 5).

From the expression z=x+y above, we can calculate the value of z to be 11.

In PHP these letters are called **variables.**

**PHP Variables**

As with algebra, PHP variables can be used to hold values (x=5) or expressions (z=x+y).

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

Rules for PHP variables:

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_)
* Variable names are case sensitive ($y and $Y are two different variables

Remember that PHP variable names are case-sensitive!

## Creating (Declaring) PHP Variables

PHP has no command for declaring a variable.

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

## 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:** When you assign a text value to a variable, put quotes around the value.

## 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.

In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

## 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 and Global Scope

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

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

The following example tests variables with local and global scope:

## Example

<?php  
$x=5; // global scope  
  
function myTest() {  
  $y=10; // local scope  
  echo "<p>Test variables inside the function:</p>";  
  echo "Variable x is: $x";  
  echo "<br>";  
  echo "Variable y is: $y";  
}   
  
myTest();  
  
echo "<p>Test variables outside the function:</p>";  
echo "Variable x is: $x";  
echo "<br>";  
echo "Variable y is: $y";  
?>

In the example above there are two variables $x and $y and a function myTest(). $x is a global variable since it is declared outside the function and $y is a local variable since it is created inside the function.

When we output the values of the two variables inside the myTest() function, it prints the value of $y as it is the locally declared, but cannot print the value of $x since it is created outside the function.

Then, when we output the values of the two variables outside the myTest() function, it prints the value of $x, but cannot print the value of $y since it is a local variable and it is created inside the myTest() function.

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 myTest() {  
  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.

The example above can be rewritten like this:

## 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.

## 4.5 PHP ARITHMETIC OPERATORS

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Operator** | **Name** | | **Example** | | **Result** | |
| + | | Addition | | $x + $y | | Sum of $x and $y | |
| - | | Subtraction | | $x - $y | | Difference of $x and $y | |
| \* | | Multiplication | | $x \* $y | | Product of $x and $y | |
| / | | Division | | $x / $y | | Quotient of $x and $y | |
| % | | Modulus | | $x % $y | | Remainder of $x divided by $y | |

The example below shows the different results of using the different arithmetic operators:

## Example

<?php   
$x=10;   
$y=6;  
echo ($x + $y); // outputs 16  
echo ($x - $y); // outputs 4  
echo ($x \* $y); // outputs 60  
echo ($x / $y); // outputs 1.6666666666667   
echo ($x % $y); // outputs 4   
?>

## PHP Assignment Operators

The PHP assignment operators is used 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Assignment** | **Same as...** | | **Description** |
| x = y | | x = y | The left operand gets set to the value of the expression on the right |
| x += y | | x = x + y | Addition |
| x -= y | | x = x - y | Subtraction |
| x \*= y | | x = x \* y | Multiplication |
| x /= y | | x = x / y | Division |
| x %= y | | x = x % y | Modulus |

The example below shows the different results of using the different assignment operators:

## Example

<?php   
$x=10;   
echo $x; // outputs 10  
  
$y=20;   
$y += 100;  
echo $y; // outputs 120  
  
$z=50;  
$z -= 25;  
echo $z; // outputs 25  
  
$i=5;  
$i \*= 6;  
echo $i; // outputs 30  
  
$j=10;  
$j /= 5;  
echo $j; // outputs 2  
  
$k=15;  
$k %= 4;  
echo $k; // outputs 3  
?>

**PHP String Operators**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | | **Example** | **Result** |
| . | | Concatenation | $txt1 = "Hello" $txt2 = $txt1 . " world!" | Now $txt2 contains "Hello world!" |
| .= | | Concatenation assignment | $txt1 = "Hello" $txt1 .= " world!" | Now $txt1 contains "Hello world!" |

The example below shows the results of using the string operators:

## Example

<?php  
$a = "Hello";  
$b = $a . " world!";  
echo $b; // outputs Hello world!   
  
$x="Hello";  
$x .= " world!";  
echo $x; // outputs Hello world!   
?>

## PHP Increment / Decrement Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **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 |

The example below shows the different results of using the different increment/decrement operators:

## Example

<?php  
$x=10;   
echo ++$x; // outputs 11  
  
$y=10;   
echo $y++; // outputs 10  
  
$z=5;  
echo --$z; // outputs 4  
  
$i=5;  
echo $i--; // outputs 5  
?>

## PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | | **Example** | **Result** |
| == | | Equal | $x == $y | True if $x is equal to $y |
| === | | Identical | $x === $y | True if $x is equal to $y, and they are of the same type |
| != | | Not equal | $x != $y | True if $x is not equal to $y |
| <> | | Not equal | $x <> $y | True if $x is not equal to $y |
| !== | | Not identical | $x !== $y | True if $x is not equal to $y, or they are not of the same type |
| > | | Greater than | $x > $y | True if $x is greater than $y |
| < | | Less than | $x < $y | True if $x is less than $y |
| >= | | Greater than or equal to | $x >= $y | True if $x is greater than or equal to $y |
| <= | | Less than or equal to | $x <= $y | True if $x is less than or equal to $y |

The example below shows the different results of using some of the comparison operators:

## Example

<?php  
$x=100;   
$y="100";  
  
var\_dump($x == $y);  
echo "<br>";  
var\_dump($x === $y);  
echo "<br>";  
var\_dump($x != $y);  
echo "<br>";  
var\_dump($x !== $y);  
echo "<br>";  
  
$a=50;  
$b=90;  
  
var\_dump($a > $b);  
echo "<br>";  
var\_dump($a < $b);  
?>

## PHP Logical Operators

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **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 |

**Rujukan /***References* **:**

[**https://www.w3schools.com/**](https://www.w3schools.com/)

[**https://www.tutorialspoint.com/**](https://www.tutorialspoint.com/)