Unit – IV

PHP

When working with data values in PHP, we need some convenient way to store these values so that we can easily access them and make reference to them whenever necessary. This is where PHP variables come in. It is often useful to think of variables as computer memory locations where data is to be stored. When declaring a variable in PHP it is assigned a name that can be used to reference it in other locations in the PHP script. The value of the variable can be accessed, the value can be changed, and the type of variable can be altered all by referencing the name assigned at variable creation time.

Naming and Creating a Variable in PHP

Before learning how to declare a variable in PHP it is first important to understand some rules about variable names (also known as *variable naming conventions*). All PHP variable names must be pre-fixed with a *$*. It is this prefix which informs the PHP pre-processor that it is dealing with a variable. The first character of the name must be either a letter or an underscore (_). The remaining characters must comprise only of letters, numbers or underscores. All other characters are deemed to be invalid for use in a variable name. Let's look at some valid and invalid PHP variable names:

```
$_myName // valid
$myName  // valid
$__myvar // valid
$myVar21 // valid
$_1Big   // invalid - underscore must be followed by a letter
$1Big    // invalid - must begin with a letter or underscore
$_er-t   // invalid contains non alphanumeric character (-)
```

Variable names in PHP are case-sensitive. This means that PHP considers *$_myVariable* to be a completely different variable to one that is named "*$_myvariable*.

Assigning a Value to a PHP Variable

Values are assigned to variables using the PHP *assignment operator*. The assignment operator is represented by the = sign. To assign a value to a variable therefore, the variable name is placed on the left of the expression, followed by the assignment operator. The value to be assigned is then placed to the right of the assignment operator. Finally the line, as with all PHP code statements, is terminated with a semi-colon (;).

Let's begin by assigning the word "Circle" to a variable named myShape:

```
$myShape = "Circle";
```

We have now declared a variable with the name *myShape* and assigned a string value to it of "Circe". We can similarly declare a variable to contain an integer value:

```
$numberOfShapes =  6;
```

The above assignment creates a variable named *numberOfShapes* and assigns it a numeric value of 6. Once a variable has been created, the value assigned to that variable can be changed at any time using the same assignment operator approach:

```
<?php
$numberOfShapes =  6; // Set initial values
$myShape = "Circle";
$numberOfShapes =  7; // Change the initial values to new values
$myShape = "Square";

?>
```

Accessing PHP Variable Values

Now that we have learned how to create a variable and assign an initial value to it we now need to look at how to access the value currently assigned to a variable. In practice, accessing a variable is as simple as referencing the name it was given when it was created.

For example, if we want to display the value which we assigned to our *numberOfShapes* variable we can simply reference it in our *echo* command:

```
<?php
echo "The number of shapes is $numberOfShapes.";
?>
```

This will cause the following output to appear in the browser:

The number of shapes is 6.

Similarly we can display the value of the *myShape* variable:

```
<?php
echo "$myShape is the value of the current shape.";
?>
```

The examples we have used for accessing variable values are straightforward because we have always had a space character after the variable name. The question arises as to what should be done if we need to put other characters immediately after the variable name. For example:

```
<?php
```

```
echo "The Circle is the $numberOfShapesth shape";
?>
```

What we are looking for in this scenario is output as follows:

The Circle is the 6th shape.

Unfortunately PHP will see the *th* on the end of the $numberOfShapes variable name as being part of the name. It will then try to output the value of a variable called $numberOfShapesth, which does not exist. This results in nothing being displayed for this variable:

The Circle is the shape.

Fortunately we can get around this issue by placing braces ({ and }) around the variable name to distinguish the name from any other trailing characters:

```
<?php
echo "The Circle is the ${numberOfShapes}th shape";
?>
```

To give us the desired output:

> The Circle is the 6th shape.

Internal (built-in) functions

PHP comes standard with many functions and constructs. There are also functions that require specific PHP extensions compiled in, otherwise fatal "undefined function" errors will appear. For example, to use image functions such as imagecreatetruecolor(), PHP must be compiled with GD support. Or, to use mysql_connect(), PHP must be compiled with MySQL support. There are many core functions that are included in every version of PHP, such as the string and variable functions. A call to phpinfo() or get_loaded_extensions() will show which extensions are loaded into PHP. Also note that many extensions are enabled by default and that the PHP manual is split up by extension. See the configuration, installation, and individual extension chapters, for information on how to set up PHP.

Reading and understanding a function's prototype is explained within the manual section titled how to read a function definition. It's important to realize what a function returns or if a function works directly on a passed in value. For example, str_replace() will return the modified string while usort() works on the actual passed in variable itself. Each manual page also has specific information for each function like information on function parameters, behavior changes, return values for both success and failure, and availability information. Knowing these important (yet often subtle) differences is crucial for writing correct PHP code.

PHP User Defined 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 in PHP

A user defined function declaration starts with the word "function":

Syntax

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

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:

Example

```
<?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 seperate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<?php
function familyName($fname) {
  echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

Example

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

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<?php
function setHeight($minheight = 50) {
  echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
```

```
setHeight(80);
?>
```

PHP Functions - Returning values

To let a function return a value, use the return statement:

Example

```
<?php
function sum($x, $y) {
  $z = $x + $y;
  return $z;
}
echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

Connecting to a Database

PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

Should I Use MySQLi or PDO?

If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, where as MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

MySQL Examples in Both MySQLi and PDO Syntax

In this, and in the following chapters we demonstrate three ways of working with PHP and MySQL: