PHP

Musadaq Mansoor

PHP

PHP is a scripting language that allows you to create dynamic web pages

You can embed php scripting within normal html coding

PHP was designed primarily for the web

PHP includes a comprehensive set of database access functions

Scripting vs Programming

A script is interpreted line by line every time it is run

A true programming language is compiled from its human readable form(source code) into a machine readable form (binary code) which is delivered to the user as a program.

Variables in scripting languages are typeless whereas variables in programs need to be declared as a particular type and have memory allocated to them.

PHP requires programming skills

PHP web sites should be developed within a software engineering framework

PHP Competitors

Perl

Microsoft Active Server Pages (ASP)

Java Server Pages (JSP)

Allaire Cold Fusion

PHP Strengths

High performance

- see benchmarks at http://www.zend.com

Interfaces to different database systems

Low cost

(Apache, PHP and MySQL) for Windows

Ease of learning and use

Portability

Basics of PHP

PHP files end with .php

PHP code is contained within tags

Canonical: <?php ?> or Short-open: <? ?>

Recommend canonical tags so as not to confuse with xml tags

Variables

All variables begin with $ and can contain letters, digits and underscore (and no digit directly after the $)

The value of a variable is the value of its most recent assignment

Don’t need to declare variables

Constant vs. Variable

The **difference between PHP Constants and Variables** are: There is no need to write a dollar sign ($) before a **constant**, where as in **Variable** one has to write a dollar sign.

Global variables aren't **constant** (you can change the value **of** a **global variable**, but you can only define a **constant** once).

Constants and Globals

To define a constant:

```
define(“PI”, 3.1416);
$area = PI*$radius*$radius ;
```

Globals:

Defined outside any function; eg form variables

```
...global $var1, $var2 …
…function xyz()
    {
      $localvarX = $var1
 …}
```

PHP Concatenation

The full stop (period or dot, to some) is used for joining text. Suppose you want to print out the following "My variable contains the value of 10". In PHP, you can do it like this:

```
<?php

$first_number = 10;

$direct_text = 'My variable contains the value of ';

echo $direct_text.$first_number;

?>
```

Operators

Operators are used to perform operations on variables and values.

Arithmetic operators

Assignment operators

Comparison operators

Increment/Decrement operators

Logical operators

String operators

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
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)

PHP 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.

PHP Assignment Operators

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

PHP Comparison Operators

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

Operator	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

PHP 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.

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

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

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

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

Arrays

You set up an array like this:

```
$name_of_array = array( );
```

First you type out what you want your array to be called (**$name_of_array**, in the array above) and, after an equals sign, you type this:

```
array( );
```

So setting up an array just involves typing the word array followed by a pair of round brackets.

Method One

The first method involves typing your values between the round brackets of array(). In the code below, we're setting up an array to hold the seasons of the year:

```
$seasons = array( "Autumn", "Winter", "Spring", "Summer" );
```

So the name of the array is $seasons. Between the round brackets of array(), we have typed some values. Each value is separated by a comma:

```
("Autumn", "Winter", "Spring", "Summer")
```

Print Array

```
print_r($name_of_array);
```

Assigning Key

The first position is always zero, unless you tell PHP otherwise. But the position is know as a Key. The Key then has a value attached to it. You can specify your own numbers for the Keys. If so, you do it like this:

```
$seasons = array( 1 => "Autumn", 2 => "Winter", 3 => "Spring", 4 => "Summer" );
```

```
$Array_Name = array(10, 20, 30, 40);
```

```
$Array_Name = array(1 => 10, 2 => 20, 3 => 30, 4 => 40);
```

Method two – Assign values to an array

Another way to put values into an array is like this:

```
$seasons = array();
$seasons[ ]="Autumn";
$seasons[ ]="Winter";
$seasons[ ]="Spring";
$seasons[ ]="Summer";
```

Method two – Assign values to an array

Here, the array is first set up with

```
$seasons = array();
```

This tells PHP that you want to create an array with the name of $seasons. To store values in the array you first type the name of the array, followed by a pair of square brackets:

```
$seasons[ ]
```

Method two – Assign values to an array

If you want different numbers for your keys, then simply type them between the square brackets:

```
$seasons[1]="Autumn";
$seasons[2]="Winter";
$seasons[3]="Spring";
$seasons[4]="Summer";
```

Function

A function is a self-contained block of code that performs a specific task and can be called by your scripts.

When called, the function's code is executed. You can pass values to functions , which they will then work with.

When finished, a function can pass a value back to the calling code.

Function

To keep the browser from executing a script when the page loads, you can put your script into a function.

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

You may call a function from anywhere within a page.

There are two types of functions

- Built-in functions
- User defined functions

However, both are called in the same way.

Function

The general form of a function call is

```
func(arg1,arg2,…)
```

The number of arguments varies from one function to another.

Each argument can be any valid expression, including other function calls.

Here is a simple example of a predefined function:

```
$length = strlen("John");
```

strlen is a standard PHP function that returns the length of a string. Therefore, $length is assigned the length of the string "John": 4.

Function

Here is an example of a function call being used as a function argument:

```
$length = strlen(strlen("John"));
```

First, the inner strlen("John") is executed, which results in the integer 4. So, the code simplifies to

```
$length = strlen(4);
```

strlen() expects a string, and therefore (due to PHP's magical autoconversion between types) converts the integer 4 to the string "4", and thus, the resulting value of $length is 1, the length of "4".

String Functions

Ucwords(string)

strtoupper(string)

strtolower(string)

trim(string)

ltrim(string)

rtrim(string)

strlen(string)

strpos(string,find)

explode(separator,string)

implode(separator,array)

Ucwords, strtoupper, strtolower

The ucwords() function converts the first character of each word in a string to uppercase.

The strtoupper() function converts all characters in a string to uppercase.

The strtolower() function converts all characters in a string to lowercase.

Trim, rtrim, ltrim, strlen

The trim() function removes whitespace from both sides of a string.

The ltrim() function removes whitespace from the left side of a string

The rtrim() function removes whitespace from the right side of a string

The strlen() function returns the length of a string on success, and 0 if the string is empty.

strpos

The strpos() function finds the position of the first occurrence of a string inside another string.

explode , implode

The explode() function breaks a string into an array.

The implode() function returns a string from the elements of an array.

User Defined Functions

Functions those are defined by user according to user requirements.

To create a new function, use the function keyword.

Syntax for declaring a function is:

```
function function-name(argument1,argument2,……)
{
code to be executed;
return statement;
}
```

Note: return statement & argument are optional.

User Defined Functions

The statements in the body of the function are executed and if any of the executed statements are return statements , the function stops execution and returns the value.

Otherwise, the function completes after the last statement is executed, without returning a value.

Functions can be defined before or after they are called.

Simple User defined function

A simple function that writes my name when it is called:

```
<?php
function writeName()
{
echo "KHAN";
}
echo "My name is ";
writeName();
?>
```

Output: My name is KHAN

Return values

A function call can be anywhere in the program, and it can be inside another function too.

- A simple function that returns value when it is called:

```
<?php
function add($x,$y)
{
$total=$x+$y;
return $total;
}
echo "1 + 16 = " . add(1,16);
?>
```

Output: 1 + 16 = 17

Note: Function will return only one value.

Function with arguments

Information may be passed to functions via the argument list, which is a comma-delimited list of expressions.

```
<?php
function calculateIncentive($salary,$incentivepercentage)
{
$incentive_amt=$salary*($incentivepercentage/100);
echo “Incentive Amount:”. $incentive_amt;
}
$salaryamt=25000; $incentive_percentage=5;
calculateIncentive($salaryamt,$incentive_percentage);
?>
```

Default Parameters

Default parameters enable you to specify a default value for function parameters that aren’t passed to the function during the function call.

The default values you specify must be a constant value and default value can be specified for each of the last arguments.

To do this, simply use the assignment operator and a value for the arguments in the function definition.

If the value is specified then this default value is ignored and the passed value is used instead.

Example

```
<?php
echo "<h3>Use of Default Parameter<br></h3>";
function increment($num, $increment = 1)
{
$num += $increment;
echo $num;
}
$num = 4;
increment($num);
increment($num, 3);
?>
```

Output: 5 7

Example

```
<?php
echo "<h3>Use of Default Parameter<br></h3>";
function increment($num, $increment)
{
$num += $increment;
echo $num;
}
$num = 4;
increment($num);
increment($num, 3);
?>
```

Output: 4 7

Dynamic Function Calls/ Function Call using Variable

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself.

```
<?php
function sayHello()
{
print "hello<br>";
}
$function_holder = "sayHello";
$function_holder();
?>
```

Output: hello

Pass by value and pass by reference

Passing a variable by reference or by value to a function is a very useful concept for PHP programmers. Normally, when you are passing a variable to a function , you are passing “by value” which mean the PHP processor will make a duplicate variable to work on. When you pass a variable to a function “by reference” you passing the actual variable and all work done on that variable will be done directly on it.

Pass by value-Example

```
<?php
$a=10; $b=20;
echo "<h3>Before Swapping<br></h3>";
echo "a:".$a."b:".$b."<br>";
swap($a,$b);
echo "<h3>After Swapping<br></h3>";
echo "a:".$a."b:".$b."<br>";
function swap($no1,$no2)
{
$temp=$no1; $no1=$no2; $no2=$temp; }
?>
```

Pass by reference-Example

```
<?php
$a=10; $b=20;
echo "<h3>Before Swapping<br></h3>";
echo "a:".$a."b:".$b."<br>";
swap($a,$b);
echo "<h3>After Swapping<br></h3>";
echo "a:".$a."b:".$b."<br>";
function swap(&$no1,&$no2)
{
$temp=$no1; $no1=$no2; $no2=$temp; }
?>
```

Function Variable Scope

Scope can be defined as the range of availability a variable has to the program in which it is declared.

Every function has its own set of variables. Any variables used outside the function's definition are not accessible from within the function by default.

When you use new variables inside a function, they are defined within the function only and don't hang around after the function call ends.

The variables that are declared outside the function are called global variables.

Example

```
<?php
function myfunction()
{
$GLOBALS["no1"]=10;
}
$no1=20;
myfunction();
echo $no1;
?>
```

Output: 10

Example

```
<?php
function add()
{
$no1=10; $no2=20;
echo $no1+$no2."<br>";
echo $no1+$GLOBALS["no2"];
}
$no2=40;
add();
?>
```

Output: 30

50

Static Variable

Like C, PHP supports declaring local function variables as static.

These kind of variables remain in tact in between function calls, but are still only accessible from within the function they are declared.

Static variables can be initialized , and this initialization only takes place the first time the static declaration is reached.

Note: The use of static variables can be fully achieved only when they are declared inside a function.

Example

```
<?php
function display()
{
Static $no=10;
echo $no;
$no++;
}
display();
display();
?>
```

Output: 10 11

Error

How to write an error handler.

Normally, it displays a message indicating the cause of the error and may also terminate script execution when a PHP script encounters an error.

Now, while this behaviour is acceptable during the development phase, it cannot continue once a PHP application has been released to actual users.

It is more professional to intercept these errors and either resolve.

Types of Errors in PHP

Notices: These are trivial, non-critical errors that PHP encounters while executing a script.

Warnings: These are more serious errors - for example, attempting to include() a file which does not exist.

Fatal errors: These are critical errors - for example, instantiating an object of a non-existent class, or calling a non-existent function.

Warning Example

```
<?php
$string = 'a string';
explode($string);
echo " code still runs ";
?>
```

Output

```
Warning: explode() expects at least 2 parameters, 1 given in
C:\xampp\htdocs\bcs\lec2004\err1.php on line 6
code still runs
```

Error Example

```
<?php

callMeJoe();

echo "code still runs";

?>
```

Output

```
Fatal error: Call to undefined function callMeJoe() in
C:\xampp\htdocs\bcs\lec2004\err2.php on line 4
```

Hide Errors

control which errors are displayed to the user (built-in PHP function called error_reporting())

this function tells the script to report only errors that match that type

"hide" non-fatal and fatal errors both

Hide Warning Example

```
<?php
// report only fatal errors
error_reporting(E_ERROR);
// initialize the $string variable
$string = 'string';
// attempt to explode() a string
// this will not generate a warning because only fatal errors are reported
explode($string);
?>
```

Output:

No output / blank screen

Hide Error Example

```
<?php
// report no fatal errors
error_reporting(~E_ERROR);
// call a non-existent function
callMeJoe();
?>
```

Output

No output

Error Reporting

Although the script above will not display a visible error message, script execution will still stop at the point of error and statements subsequent to that point will not be executed.

error_reporting() gives you control over which errors are displayed; it doesn't prevent the errors themselves.

Rolling Your Own

Changing the way errors are handled.

Function called set_error_handler(), it allows to divert all PHP errors to a custom function that are defined, instead of sending them to the default handler.

This custom function must be capable of accepting a minimum of two mandatory arguments:

- error type
- corresponding descriptive message
- file name (optional)
- line number (optional)
- Context (optional)

Custom Error Handler

```
<?php

set_error_handler('oops');

$string = 'a string';

explode($string);

//custom function

function oops($type, $msg, $file, $line, $context) {

echo "<h1>Error!</h1>";

echo "An error occurred while executing this script. Please contact the <a
href=mailto:webmaster@somedomain.com>webmaster</a> to report this error.";

echo "<p />";

echo "Here is the information provided by the script:";

echo "<hr><pre>";
```

Custom Error Handler

```
echo "Error code: $type<br />";
echo "Error message: $msg<br />";
echo "Script name and line number of error: $file:$line<br />";
$variable_state = array_pop($context);
echo "Variable state when error occurred: ";
print_r($variable_state);
echo "</pre><hr>";
}
?>
```

Output : Shown by running the code.

Advantage

These arguments are then used to create an error page that is friendlier and more informative than PHP's standard one-line error message

Pulling the Trigger

PHP allows you to use its built-in error handling system to raise your own custom errors as well.

This is accomplished via a function named trigger_error(), which allows you to raise any of the three error types reserved for users: E_USER_NOTICE, E_USER_WARNING and E_USER_ERROR.

When these errors are triggered, PHP's built-in handler will automatically wake up to handle them.

Example

```
<?php
function testNumber($num) {
if (is_float($num)) {
trigger_error("Number $num is not an integer", E_USER_WARNING);
}
if ($num < 0) {
trigger_error("Number $num is negative", E_USER_ERROR);
} }
// test the function with different values
testNumber(100);
testNumber(5.6);
echo "code still runs";
testNumber(-8);
echo "code still runs";
?>
```

Output

Warning: Number 5.6 is not an integer in C:\xampp\htdocs\bcs\lec2004\err6.php on line 9

Fatal error: Number -8 is negative in C:\xampp\htdocs\bcs\lec2004\err6.php on line 1

Exception Handling

Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

The current code state is saved.

The code execution will switch to a predefined (custom) exception handler function.

Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code.

Basic use of Exception

When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.

If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

Basic use of Exception

Lets try to throw an exception without catching it:

```
<?php
//create function with an exception
function checkNum($number) {
if($number>1) {
throw new Exception("Value must be 1 or below");}
return true;
}
//trigger exception
checkNum(2);
?>
```

Output

Fatal error: Uncaught exception 'Exception' with message 'Value must be 1 or below' in C:\xampp\htdocs\bcs\lec2004\exc1.php:6 Stack trace: #0 C:\xampp\htdocs\bcs\lec2004\exc1.php(9): checkNum(2) #1 {main} thrown in C:\xampp\htdocs\bcs\lec2004\exc1.php on line 6

Valid Example

```
<?php
function checkNum($number) {
if($number>1) { throw new Exception("Value must be 1 or below"); }
return true;
}
try { checkNum(2); }
catch(Exception $e)
{
echo 'Message: ' .$e->getMessage();
}
?>
```

Output

Message: Value must be 1 or below

Basic HTML Form

```
<html>
<head>
<title>A BASIC HTML FORM</title>
</head>
<body>

<FORM NAME ="form1" METHOD =" " ACTION = "">

<INPUT TYPE = "TEXT">
<INPUT TYPE = "Submit" Name = "Submit1" VALUE = "Login">

</FORM>

</body>
</html>
```

Method, Action and Submit

The Method attribute is used to tell the browser how the form information should be sent.

GET

POST

The Action attribute is crucial. It means, "Where do you want the form sent?". If you miss it out, your form won't get sent anywhere.

The HTML Submit button is used to submit form data to the script mentioned in the ACTION attribute.

PHP and Text boxes

To get at the text that a user entered into a text box, the text box needs a NAME attribute.

You then tell PHP the NAME of the textbox you want to work with.

<input type = "text" name = "username">

The NAME of our textbox is **username**. It's this name that we will be using in a PHP script.

PHP and Text boxes

To return data from a HTML form element, you use the following syntax:

```
$_methodname['formElement_name'];
```

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

So you begin with a dollar sign (**$**) and an underscore character (_). Next comes the method you want to use, POST or GET. You need to type a pair of square brackets next. In between the square brackets, you type the NAME of your HTML form element

The **$_POST[ ]** is an inbuilt function you can use to get POST data from a form. If you had METHOD = "GET" on your form, then you'd used this instead:

```
$username = $_GET['username'];
```

Forms and Conditional logic

At the moment, all we're doing is returning what the user entered and printing it to the page. But we can use a bit of Conditional Logic to test what is inside of the variable. As an example, change your PHP to this:

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

if ($username == "letmein") {

print ("Welcome back, friend!");

}
else {

print ("You're not a member of this site");

}
```

We're now checking to see if the user entered the text "letmein". If so, the username is correct; if not, print another message.