PHP

PHP Hypertext preprocessor

Information Technology Institute

COURSE MATERIALS

You can access the course materials via this link

http://goo.gl/ev41na

CONTENTS-DAY01

- History of PHP.
- Why PHP?
- What do we need? (LAMP Overview)
- Installing LAMP
- PHP Overview (Variables, Constants, Flow control, ….)

HISTORY

1994

PHP originally stood for "personal home page". PHP development began by the Danish/Greenlandic programmer Rasmus Lerdorf

1997

Zeev Suraski and Andi Gutmans, two Israeli developers at the Technion IIT, rewrote the parser and formed the base of PHP 3, changed the name to PHP: Hypertext Preprocessor.

1999

They started a new rewrite of PHP's core, producing the Zend Engine, They also founded Zend Technologies

WHY PHP?

- Ease of Learning PHP.
- Object-Oriented Support
- Portability
- Source Code
- Availability of Support and Documentation

WHAT DO WE NEED?

- **LAMP** is an acronym for a solution stack of free, open source software, originally coined from the first letters of Linux (operating system), Apache HTTP Server, MySQL (database software) and a Programming language like Perl/PHP/Python, principal components to build a viable general purpose web server.

INSTALLATION

- Ubuntu:

```
$ sudo tasksel install lamp-server
```

- CentOS:

```
$ sudo yum install httpd mariadb-server
mariadb php php-mysql
```

EMBEDDING PHP IN HTML

- Simply you can PHP in HTML page by Adding the php tag as the following:

```
<html>
<body>
<?php
echo '<h1>Hello, World!</h1>';
?>
</body>
</html>
```

- The PHP interpreter will run through the script and replace it with the output from the script.

PHP IS A SERVER SIDE

- The PHP has been interpreted and executed on the web server, as distinct from JavaScript and other client-side technologies interpreted and executed within a web browser on a user's machine.
- The code that you now have in this file consists of four types of text:
  - HTML
  - PHP tags
  - PHP statements
  - Whitespace

  You can also add comments.

PHP TAGS

- XML style

```
<?php echo '<p>Hello!.</p>'; ?>
```

- Short style

```
<? echo '<p>Hello!</p>'; ?>
```

- SCRIPT style

```
<script language='php'> echo
  '<p>Hello!.</p>'; </script>
```

- ASP style

```
<% echo '<p>Hello!.</p>'; %>
```

PHP TAGS

- Using XML style is recommended because it can’t be closed off by the administrator beside it’s portable through systems.
- Short Style is the simplest and follows the style of a Standard Generalized Markup Language (SGML) processing instruction. To use this type you need to enable the `short_open_tag` setting in your config file.
- Script Style This tag style is the longest and will be familiar if you’ve used JavaScript or VBScript.
- ASP Style is the same as used in Active Server Pages (ASP) or ASP.NET. You can use it if you have enabled the `asp_tags` configuration setting in php.ini.

PHP STATEMENTS & WHITESPACES

```
echo '<p>Hello, World!.</p>';
```

- Consists of reserved word to display content in browser, each line ends with (;)
- Spacing characters such as newlines (carriage returns), spaces, and tabs are known as whitespace. As you probably already know, browsers ignore whitespace in HTML. So does the PHP engine.
  - `echo 'hello ';`
  - `echo 'world';`

  and

  - `echo 'hello ';echo 'world';`
- are equivalent, but the first version is easier to read.

COMMENTS

- C-style, multiline comment that might appear at the start of a PHP script:

```
/* Author: Islam Askar
Last modified: June 24
This is to test comments!
*/
```

- You can also use single-line comments, either in the C++ style:

```
echo ‘<p>Hello.</p>’; // Comment
```

or in the shell script style:

```
echo ‘<p>Hello.</p>’; # Comment
```

ADDING DYNAMIC CONTENT

- We will put a function to print the Date and time of the machine

```
<?php
echo "<p>Now, It’s ";
echo date('H:i, jS F Y');
echo "</p>";
?>
```

ACCESSING FORM VARIABLES

- You may be able to access the contents of the field in the following ways:
  - `$field_name` // short style
  - `$_POST['field_name']` // medium style
  - `$HTTP_POST_VARS['field_name']` // long style
- Short style ($ field_name) is convenient but requires the `register_globals` configuration setting be turned on.
- Medium style involves retrieving form variables from one of the arrays `$_POST`, `$_GET`, or `$_REQUEST`.

ACCESSING FORM VARIABLES

- Creating short variables name is recommended

```
<?php
// create short variable names
$field = $_POST['field'];
$field = $_GET['field'];
$field = $_REQUEST['field'];
?>
```

VARIABLES AND LITERALS

- Value itself is a literal.
- There are two kinds of strings:
  - Double quotation
  - Single quotation.
- PHP tries to evaluate strings in double quotation marks, resulting in the behavior shown earlier. Single-quoted strings are treated as true literals.

VARIABLES AND LITERALS

- There is also a third way of specifying strings using the heredoc syntax.
- Heredoc syntax allows you to specify long strings tidily, by specifying an end marker that will be used to terminate the string.

```
echo <<<theEnd
line 1
line 2
line 3
theEnd
```

UNDERSTANDING IDENTIFIERS

- Identifiers are the names of variables . You need to be aware of the simple rules defining valid identifiers:
  - Identifiers can be of any length and can consist of letters, numbers, and under-scores.
  - Identifiers cannot begin with a digit.
  - In PHP, identifiers are case sensitive. `$field` is not the same as `$Field` Trying to use them interchangeably is a common programming error. Function names are an exception to this rule: Their names can be used in any case.
  - A variable can have the same name as a function. This usage is confusing, however, and should be avoided. Also, you cannot create a function with the same name as another function.

EXAMINING VARIABLE TYPES

- A variable's type refers to the kind of data stored in it.
- PHP supports the following basic data types:
  - Integer—Used for whole numbers
  - Float (also called double)—Used for real numbers
  - String—Used for strings of characters
  - Boolean—Used for true or false values
  - Array—Used to store multiple data items
  - Object—Used for storing instances of classes

EXAMINING VARIABLE TYPES

- PHP is called weakly typed, or dynamically typed language. The type of a variable is determined by the value assigned to it.
- For example, when you created `$var1` and `$var2`, their initial types were determined as follows:
  - `$var1= 0;`
  - `$var2 = 0.00;`
- Strangely enough, you could now add a line to your script as follows:
  - `$var2 = ‘Hello’;`

EXAMINING VARIABLE TYPES

- You can pretend that a variable or value is of a different type by using a type cast. You simply put the temporary type in parentheses in front of the variable you want to cast.
- For example, you could have declared the two variables from the preceding section using a cast:
  - `$var1 = 0;`
  - `$var2 = (float)$var1;`

EXAMINING VARIABLE TYPES

- PHP provides one other type of variable: the variable variable.
- Variable variables enable you to change the name of a variable dynamically.
- For example, you could set

```
$varname = 'var1';
```

- You can then use `$$varname` in place of `$var1`. For example, you can set the value of `$var1` as follows:

```
$$varname = 5;
```

- This is exactly equivalent to

```
$var1= 5;
```

DECLARING CONSTANTS

- You can define these constants using the define function:

```
define('CONST1', 100);
```

- One important difference between constants and variables is that when you refer to a constant, it does not have a dollar sign in front of it. If you want to use the value of a constant, use its name only.

```
echo CONST1;
```

VARIABLE SCOPE

- The term scope refers to the places within a script where a particular variable is visible.
- The **six** basic scope rules in PHP are as follows:
  - Built-in superglobal variables are visible everywhere within a script.
  - Constants, once declared, are always visible globally; that is, they can be used inside and outside functions.
  - Global variables declared in a script are visible throughout that script, but not inside functions.
  - Global Variables inside functions refer to the global variables of the same name.
  - Static variables created inside functions are invisible from outside the function but keep their value between one execution of the function and the next.
  - Variables created inside functions are local to the function and cease to exist when the function terminates.

VARIABLE SCOPE

- Superglobals or autoglobals and can be seen everywhere, both inside and outside functions.
- The complete list of superglobals is as follows:
  - `$GLOBALS`—An array of all global variables (Like the global keyword, this allows you to access global variables inside a function—for example, as `$GLOBALS['myvariable']`.)
  - `$_SERVER`—An array of server environment variables
  - `$_GET`—An array of variables passed to the script via the GET method.
  - `$_POST`—An array of variables passed to the script via the POST method.
  - `$_REQUEST`—An array of all user input including the contents of input including $_GET, $_POST & $_COOKIE (but not $_FILES since PHP 4.3.0).
  - `$_COOKIE`—An array of cookie variables
  - `$_FILES`—An array of variables related to file uploads
  - `$_ENV`—An array of environment variables
  - `$_SESSION`—An array of session variables

OPERATORS AND PRECEDENCE

- Arithmetic operators are straightforward; they are just the normal mathematical operators.

Operator	Name	Example
+	Addition	$a + $b
-	Subtraction	$a - $b
*	Multiplication	$a * $b
/	Division	$a / $b
%	Modulus	$a % $b

- With each of these operators, you can store the result of the operation, as in this example:

```
$result = $a + $b;
```

OPERATORS AND PRECEDENCE

- You can use the string concatenation operator to add two strings and to generate and store a result much as you would use the addition operator to add two numbers:

```
$a = “Hello, “;
$b = “World!”;
$result = $a.$b;
```

- The $result variable now contains the string “Hello, World!”

OPERATORS AND PRECEDENCE

- Combined assignment operators exist for each of the arithmetic operators and for the string concatenation operator

Operator	Use	Equivalent To
+=	$a += $b	$a=$a + $b
-=	$a -= $b	$a=$a - $b
*=	$a * =$b	$a=$a * $b
/=	$a / =$b	$a=$a / $b
%=	$a % =$b	$a=$a % $b
.=	$a.=$b	$a=$a.$b

OPERATORS AND PRECEDENCE

- The pre- and post-increment (++) and decrement (--) operators are similar to the +=and -= operators, but with a couple of twists.

```
$a=4;
echo ++$a;      //echo 5 , value of $a = 5


$a=4;
echo $a++;     //echo 4 , value of $a = 5
```

OPERATORS AND PRECEDENCE

- The reference operator (&, an ampersand) can be used in conjunction with assignment.

```
$a = 5;
$b = $a;
```

- These code lines make a second copy of the value in $a and store it in $b. If you subsequently change the value of $a, $b will not change:

```
$a = 7;  // $b will still be 5
```

- You can avoid making a copy by using the reference operator. For example,

```
$a = 5;
$b = &$a;
$a = 7;  // $a and $b are now both 7
```

OPERATORS AND PRECEDENCE

- References can be a bit tricky. Remember that a reference is like an **alias** rather than like a **pointer**. Both $a and $b point to the same piece of memory. You can change this by unsetting one of them as follows:

```
unset($a);
```

- Unsetting does not change the value of $b (7) but does break the link between $a and the value 7 stored in memory.

OPERATORS AND PRECEDENCE

- The comparison operators compare two values. Expressions using these operators return either of true or false.

Operator	Name	Use
==	Equals	$a == $b
===	Identical	$a === $b
!=	Not equal	$a != $b
!==	Not identical	$a !== $b
<>	Not equal	$a <> $b
<	Less than	$a < $b
>	Greater than	$a > $b
<= , >=	Less/greater than or equal to	$a <= $b

OPERATORS AND PRECEDENCE

- The logical operators combine the results of logical conditions. $a, is between 0 and 100. using the AND operator, as follows:

```
$a >= 0 && $a <=100
```

Operator	Name	Use	Result
!	NOT	!$b	Returns true if $b is false and vice versa
&&	AND	$a && $b	Returns true if both $a and $b are true; other-wise false
||	OR	$a || $b	Returns true if either $a or $b or both are true; otherwise false
and	AND	$a and $b	Same as &&, but with lower precedence
or	OR	$a or $b	Same as ||, but with lower precedence
xor	XOR	$a x or $b	Returns true if either $a or $b is true, and false if they are both true or both false.

OPERATORS AND PRECEDENCE

- The comma operator (,) separates function arguments and other lists of items. It is normally used incidentally.
- Two special operators, new and ->, are used to instantiate a class and access class members, respectively.
- The ternary operator (?:) takes the following form:

```
condition ? value if true : value if false
```

- This operator is similar to the expression version of an if-else statement, A simple example is

```
($grade >= 50 ? 'Passed' : 'Failed')
```

OPERATORS AND PRECEDENCE

- The error suppression operator (@) can be used in front of any expression—that is, any-thing that generates or has a value. For example,

```
$a = @(57/0);
```

- Without the @ operator, this line generates a divide-by-zero warning. With the operator included, the error is suppressed.
- The execution operator is really a pair of operators—a pair of backticks (``) in fact. The backtick is not a single quotation mark; it is usually located on the same key as the ~ (tilde) symbol on your keyboard.

```
$out = `ls -la`;

echo '<pre>'.$out.'</pre>';
```

OPERATORS AND PRECEDENCE

- There are a number of array operators. The array element operators ([]) enables you to access array elements.

Operator	Name	Use	Result
+	Union	$a+$b	Returns an array containing everything in $a and $b
==	Equality	$a == $b	Returns true if $a and $b have the same key and pairs
===	Identity	$a === $b	Returns true if $a and $b have the key and value pairs the same order
!=	Inequality	$a and $b	Returns true if $a and $b are not equal
<>	Inequality	$a or $b	Returns true if $a and $b are not equal
!==	Non-identity	$a x or $b	Returns true if $a and $b are not identical

OPERATORS AND PRECEDENCE

- There is one type operator: instanceof. This operator is used in object-oriented programming.
- The instanceof operator allows you to check whether an object is an instance of a particular class, as in this example:

```
class sampleClass{};
$myObject = new sampleClass();
if ($myObject instanceof sampleClass)
echo “myObject is an instance of
   sampleClass”;
```

OPERATORS AND PRECEDENCE

Associativity	Operators
Left	,
Left	Or
Left	Xor
Left	And
Right	Print
Left	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
Left	: ?
Left	||
Left	&&
Left	|
Left	^
Left	&

OPERATORS AND PRECEDENCE

Associativity	Operators
n/a	== != === !==
n/a	< <= > >=
Left	<< >>
Left	+ - .
Left	* / %
Right	! ~ ++ -- (int) (double) (string) (array) (object) @
Right	[]
n/a	New
n/a	()

VARIABLE FUNCTIONS

- To use `gettype()`, you pass it a variable. It determines the type and returns a string containing the type name: bool, int, double (for floats), string, array, object, resource, or NULL. It returns unknown type if it is not one of the standard types.

```
string gettype(mixed var);
```

- `settype()`, you pass it a variable for which you want to change the type and a string containing the new type for that variable from the previous list.

```
bool settype(mixed var, string type);
```

VARIABLE FUNCTIONS

- `is_array()`—Checks whether the variable is an array.
- `is_double()`, `is_float()`, `is_real()` (All the same function)—Checks whether the variable is a float.
- `is_long()`, `is_int()`, `is_integer()` (All the same function)—Checks whether the variable is an integer.
- `is_string()`—Checks whether the variable is a string.
- `is_bool()`—Checks whether the variable is a boolean.

VARIABLE FUNCTIONS

- `is_object()`—Checks whether the variable is an object.
- `is_resource()`—Checks whether the variable is a resource.
- `is_null()`—Checks whether the variable is null.
- `is_scalar()`—Checks whether the variable is a scalar, that is, an integer, boolean, string, or float.
- `is_numeric()`—Checks whether the variable is any kind of number or a numericstring.
- `is_callable()`—Checks whether the variable is the name of a valid function.

VARIABLE FUNCTIONS

- `isset()` function takes a variable name as an argument and returns true if it exists and false otherwise. You can also pass in a comma-separated list of variables, and isset() will return true if all the variables are set.

```
bool isset(mixed var);[;mixed var[,...]])
```

- You can wipe a variable out of existence by using its companion function, unset(), which has the following prototype:

```
void unset(mixed var);[;mixed var[,...]])
```

VARIABLE FUNCTIONS

- `empty()` function checks to see whether a variable exists and has a nonempty, nonzero value; it returns true or false accordingly. It has the following prototype:

```
bool empty(mixed var);
```

FLOW CONTROL

- If – else – elseif

```
if (condition) {
  statement;
} elseif (condition) {
  statement;
} elseif (condition) {
  statement;
} else
{if (condition) {
  Statement;}
}
```

FLOW CONTROL

- Switch

```
switch($var) {
    case “value" :
      Statement;
      break;
    case " value " :
      Statement;
      break;
    default :
      Statement;
      break;
}
```

FLOW CONTROL

- While Loops

```
while( condition ) expression;
```

The following while loop will display the numbers from 1 to 5:

```
$num = 1;
while ($num <= 5 ){
echo $num.”<br />”;
$num++;
}
```

FLOW CONTROL

- for and foreach Loops

```
for( expression1; condition; expression2)
expression3;
```

  - expression1 is executed once at the start. Here, you usually set the initial value of a counter.
  - The condition expression is tested before each iteration. If the expression returns false, iteration stops. Here, you usually test the counter against a limit.
  - expression2 is executed at the end of each iteration. Here, you usually adjust the value of the counter.
  - expression3 is executed once per iteration. This expression is usually a block of code and contains the bulk of the loop code.

FLOW CONTROL

- do...while Loops

```
do
expression;
    while( condition );
```

- Example:

```
$num = 100;
do{
echo $num."<br />";
}while ($num < 1 ) ;
```

FLOW CONTROL

- use the `break` statement in a loop, execution of the script will continue at the next line of the script after the loop.
- If you want to jump to the next loop iteration, you can instead use the `continue` statement.
- If you want to finish executing the entire PHP script, you can use `exit`. This approach is typically useful when you are performing error checking

FLOW CONTROL

- For all the control structures we have looked at, there is an alternative form of syntax. It consists of replacing the opening brace ({) with a colon (:) and the closing brace with a new keyword, which will be `endif, endswitch, endwhile, endfor, or endforeach`, depending on which control structure is being used. No alternative syntax is available for do...while loops.