Week 2
Week 2 Tasks

1. Assignment: Read Topic: The Building Blocks: Data Types, Variables, Constants, Operators and Expressions

(Note: Chapter 5 in "Sams Teach Yourself PHP, MySQL, and Apache")
2. Read "Additional Reading" materials (please scroll down the page).
3. Visit the following reference sites to get familiar with HTML:

http://www.w3schools.com/html/

http://www.davesite.com/webstation/html/

http://www.echoecho.com/html.htm

http://www.cwru.edu/help/introHTML/TCh1.html

http://www.htmlcodetutorial.com/

http://www.mcli.dist.maricopa.edu/tut/lessons.html

http://www.w3.org/MarkUp/Guide/

http://www.htmlgoodies.com/primers/html/article.php/3478131

Please note that you can refer to the above HTML websites throughout the course.

4. Lab Work: For further instructions on how to submit lab work, click "Tasks, Tests and Surveys" "Lab 1."

Lecture Notes:

Variables

In PHP, all variable names begin with a dollar sign ($). The $ is followed by an alphabetic character or an underscore, and optionally followed by a sequence of alphanumeric characters and underscores. There is no limit on the length of a variable. Variable names in PHP are case-sensitive. Here are some examples:

```
$i
$counter
$first_name
$_TMP
```

In PHP, unlike in many other languages, you do not have to explicitly declare variables. PHP automatically declares a variable the first time a value is assigned to it. PHP variables are untyped; you can assign a value of any type to a variable.

Dynamic Variables

Sometimes it is useful to set and use variables dynamically. Normally, you assign a variable like this:

```
$var = "hello";
```

Now let's say you want a variable whose name is the value of the $var variable. You can do that like this:

```
$$var = "World";
```

PHP parses $$var by first dereferencing the innermost variable, meaning that $var becomes "hello". The expression that is left is then $"hello", which is just $hello. In other words, we have just created a new variable named hello and assigned it the value "World". You can nest dynamic variables to an infinite level in PHP, although once you get beyond two levels, it can be very confusing for someone who is trying to read your code.

There is a special syntax for using dynamic variables inside quoted strings in PHP:

```
echo "Hello ${$var}";
```

This syntax is also used to help resolve an ambiguity that occurs when variable arrays are used. Something like $$var[1] is ambiguous because it is impossible for PHP to know which level to apply the array index to. ${$var[1]} tells PHP to dereference the inner level first and apply the array index to the result before dereferencing the outer level. ${$var}[1], on the other hand, tells PHP to apply the index to the outer level.

Dynamic variables may not initially seem that useful, but there are times when they can shorten the amount of code you need to write to perform certain tasks. For example, say you have an associative array that looks like this:

```
$array["abc"] = "Hello";
```

```
$array["def"] = "World";
```

Associative arrays like this are returned by various functions in the PHP modules. mysql_fetch_array( ) is one example. The indices in the array usually refer to fields or entity names within the context of the module you are working with. It can be handy to turn these entity names into real PHP variables, so you can refer to them as simply $abc and $def. This can be done as follows:

```
while(list($index,$value) = each($array)) {
  $$index = $value;
}
```

Data Types

PHP provides three primitive data types: integers, floating point numbers, and strings. In addition, there are two compound data types: arrays and objects.

Integers

Integers are whole numbers. The range of integers in PHP is equivalent to the range of the long data type in C. On 32-bit platforms, integer values can range from -2,147,483,648 to +2,147,483,647. PHP automatically converts larger values to floating point numbers if you happen to overflow the range. An integer can be expressed in decimal (base-10), hexadecimal (base-16), or octal (base-8). For example:

```
$decimal=16;
$hex=0x10;
$octal=020;
```

Floating Point Numbers

Floating point numbers represent decimal values. The range of floating point numbers in PHP is equivalent to the range of the double type in C. On most platforms a double can range from 1.7E-308 to 1.7E+308. A double may be expressed either as a regular number with a decimal point or in scientific notation. For example:

```
$var=0.017;
$var=17.0E-3
```

Note that PHP also has a set of functions known as the BC (binary calculator) functions. These functions can manipulate arbitrary precision numbers. If you are dealing with very large numbers or numbers that require a high degree of precision, you should use these functions.

Strings

A string is a sequence of characters. A string can be delimited by single quotes or double quotes:

```
'PHP is cool'
"Hello, World!"
```

Double-quoted strings are subject to variable substitution and escape sequence handling, while single quotes are not. For example:

```
$a="World";
echo "Hello\t$a\n";
```

This displays "Hello" followed by a tab and then "World" followed by a newline. In other words, variable substitution is performed on the variable $a and the escape sequences are converted to their corresponding characters. Contrast that with:

```
echo 'Hello\t$a\n';
```

In this case, the output is exactly "Hello\t$a\n". There is no variable substitution or handling of escape sequences.

The following table shows the escape sequences understood by PHP:

Escape Sequence	Meaning
\n	Newline
\t	Tab
\r	Carriage return
\\	Backslash
\$	Dollar sign

****

Variables:

Variables are assigned in a typical manner. Normal variables are preceded with a $, and assigned to with =. Any variable can be used as an array by including a key in square brackets after the variable. For a regular array, a number is used as the key. For an associative array, a quotes enclosed string (or another variable) is used. The function "array" can also be used to create arrays of either type. The code below illustrates these features (lines beginning with "//" are comments):

```
<?php
// Normal variable assignment
```

```
$person = "Bob";

// Assigning to a regular array
$fruit[0] = "apple";
$fruit[1] = "grape";
$fruit[] = "orange"; // see explanation below

// Assigning to an associative array
$mother['kitten'] = "cat";
$mother['puppy'] = "dog";

// Using the array function for a regular array
$fruit = array('apple','grape','orange');

// Using the array function for an associative array
$mother = array('kitten' => 'cat', 'puppy' => 'dog');

// The following line prints "Bob's cat ate an orange."

$cat = $mother['kitten'] ;

echo "$person's $cat ate an $fruit[2].";

?>
```

In the sixth line above, "orange" is assigned to the regular array $fruit with no index given. This tells PHP to assign it the next available number, and thus it becomes assigned to $fruit[2]. One of the most powerful features of PHP is its easy access to form variables. For instance, consider the following short form:

```
<form method=post action="form_handler.php3">
<input type=text name="formvar">
<input type=submit>
```

On submitting the form, the value of the variable "formvar" will be available to the form_handler.php3 script simply as $formvar with no parsing required on your part. This is true for both POST and GET form calls. The form_handler.php3 will also have easy access to all environmental variables in the same manner (the REMOTE_HOST variable can be accessed as $REMOTE_HOST, for instance). This is true of all calls to PHP scripts, not just ones that were originated from a form.

Arithmetic Operators:

Arithmetic operators in PHP are virtually identical to those in Perl, as illustrated by the example code below:

```
<?php
// Addition
$ADD = $VAR + 3;

// Subtraction
$SUB = 18 - 6;

// Multiplication
$MULT = $VAR1 * $VAR2;

// Division
$DIV = 15 / $VAR;

// Modulus (Division to get remainder)
$MOD = $VAR % 2;
?>
```

****

PHP Operators

This section lists the different operators used in PHP.

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y

*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Difference between '==' (equal) and '===' (identical) comparison operators in PHP (with examples)

Two of the many comparison operators used by PHP are '==' (i.e. equal) and '===' (i.e. identical). The difference between the two is that '==' should be used to check if the values of the two operands are equal or not. On the other hand, '===' checks the values as well as the type of operands.

Let me explain more using some examples:

'==' (Equal):

```
if("22" == 22) echo "YES";
else        echo "NO";
```

The code above will print "YES". The reason is that the values of the operands are equal. Whereas when we run the example code below:

'===' (Identical):

```
if("22" === 22) echo "YES";
else         echo "NO";
```

The result we get is "NO". The reason is that although values of both operands are same their types are different, "22" (with quotes) is a string while 22 (w/o quotes) is an integer. But if we change the code above to the following:

```
if("22" === (string)22) echo "YES";
else        echo "NO";
```

Then, the result will be "YES". Notice that we changed the type of right operand to a string which is the same as the left operand (i.e. string). Now, the types and values of both left and right operands are the same hence both operands are identical.

Additional Reference:

http://www.homeandlearn.co.uk/php/php3p2.html

****

Using Variables and Constants

You explore values stored in a computer program; these values are known as variables. Variables use names, commonly known as identifiers, which must be named according to coding conventions.

Displaying Variables

You use the echo() statement to print variables. You explore the differences of using quotation marks with text strings. You can display variable names with the echo() statement.

Modifying Variables

You can change the value of a variable in a script. You use the “Hello World” script to examine how you can change values. You can integrate PHP scripts with HTML.

Defining Constants

Unlike variables, the information in constants never changes. You define constants with UPPERCASE lettering and no dollar sign ($) prefixed to the name. You learn how PHP handles constants and variables differently, by substituting one for the other.

Working with Data Types

Data types help distinguish variables and what kind they are, whether text strings, numbers, or names. Like XML, PHP defines data types for memory allocation and what kind of operation PHP can perform on a variable.

Primitive data types are perhaps more easily understood, as they define values only once. By contrast, reference, or composite, data types contain multiple values.

Numeric Data Types

Integers and floating-point numbers are specifically relevant to programming for calculations. These two numeric data types lend themselves to programming particularly well; exponential, or scientific, notation provides a shorthand for writing large floating-point numbers.

Boolean Values

Boolean values are binary; that is, they can be either true (1) or false (0). PHP, and other programming languages, use Boolean to determine whether a variable is “on” or “off.”

Building Expressions

PHP uses expressions to produce results. Expressions can contain operands like a literal value (text string or number) or operator (symbols used to manipulate operands). Binary operators and unary operators have differing notation requirements.

Arithmetic Operators

The five arithmetic binary operators affect operands in arithmetic calculations. In such calculations, note how PHP attempts to change string values to numbers.

Arithmetic Unary Operators

Increments (++) and decrements (--) are unary operators that can be used as prefix operators or postfix operators. Note how the placement of these operators affect the values returned.

Assignment Operators

Used for assigning values to variables, assignment operators can be as simple as the equal sign (=) or compound (+=, -=, *=, /=, %=). Each performs a unique mathematical calculation. You use the script on page 136 to review the various calculations that can be performed.

Comparison and Conditional Operators

You use comparison operators to compare two operands, with a Boolean value of true or false returned. You use conditional operators to execute an expression based on the result of a conditional expression.

Logical Operators

Like comparison operators, logical operators return a Boolean value of true or false, but logical operators compare operands for equality. Note how you can use the name as well as the operator for && (and) and || (or).

Type Casting

Type casting, or casting, ensures that a data type is the kind a script expects. This is helpful for form input, verifying that data is indeed of the kind the script needs to execute.

You use the gettype() function to view a variable's data type, and other functions to determine whether a variable is of a specific data type.
Resource:
http://us.php.net/gettype
http://us.php.net/manual/en/function.settype.php

Understanding Operator Precedence

Operator precedence dictates the order in which operations are evaluated. Associativity, however, takes