S@ifur's Solutions

69/B Monwara Plaza (Level-5)
East Panthapath
Dhaka-1205
Cell: 01197 11 82 77

Lecture Sheet on "PHP+MySQL"
Day-4: (PHP)

PHP Structure & Syntax

PHP programs are written using a text editor, such as Notepad or WordPad, just like HTML pages. However, PHP pages, for the most part, end in a **.php** extension. This extension signifies to the server that it needs to parse the PHP code before sending the resulting HTML code the viewer's web browser.

In a five-star restaurant, patrons see just a plate full of beautiful food served up just for them. They don't see where the foods comes from, nor how it was prepared. In a similar fashion, PHP fits right into your HTML code and is invisible to the people visiting your site.

How PHP fits with HTML

We assume that you know some HTML before you embark on your PHP/Apache/MySQL journey, and you've undoubtedly seen how JavaScript code and other languages can be interspersed within the HTML code in an HTML page. What makes PHP so different is that it not only allows HTML pages to be created on the fly; it is invisible to your web site visitors. The only thing they see when they view the source of your code is the resulting HTML output. This gives you more security for your PHP code and more flexibility in writing it.

HTML can also be written inside the PHP section of your page; this allows you to format text while keeping blocks of code together. This will also help you write organized, efficient code, and the browser (and, more important, the viewer) won't know the difference.

PHP can also be written as a standalone program, with no HTML at all. This is helpful for storing your connection variables, redirecting your visitors to another page of your site, or performing other functions that we will discuss later.

Rules of PHP Syntax

One of the benefits of using PHP is that it is relatively simple and straightforward. As with any computer language, there is usually more than one way to perform the same function. Once you feel comfortable writing some PHP programs, you can research shortcuts to make yourself and your code more efficient. For the sake of simplicity, we cover only the common uses, rules, and functions of PHP.

You should always keep in mind these two basic rules of PHP.

⇒ PHP is denoted in the page with opening and closing tags as follows:

```
<?php
?>
```

⇒ PHP lines end with a semicolon, generally speaking:

```
<?php
//First line of code goes here;
//Second line of code goes here;
//Third line of code goes here;
?>
```

Comments can be added into your program as we just did through double slashes (//) for one-liners or /*and*/ for opening and closing comment tags that may extend over several

lines of code. Indents don't matter, and, generally speaking, neither do line returns. This gives you freedom as a programmer, but a little freedom can be a dangerous thing, as we discuss in the next section.

And there you have it! Now you're an expert. Okay – there might be a few more things you need to learn, but this gets you started.

What makes a great program?

Truly professional code follows three general guidelines:

- **Consistency:** Blocks of well written code always look the same and have the same indents and ways of coding, such as syntax shortcuts that use bracket placement and formatting styles consistently throughout the program. The great thing about PHP is that it really doesn't care about tabs or indents, so you are free to create a style all your own, one that works best for you. In addition, while there may be more than one syntax for accomplishing the same goal, good coders will be consistent throughout their code with whichever method they choose. For example, as far as PHP is concerned, the following two snippets of code mean the same thing:

```
<?php
//php code goes here;
?>
<?
//php code goes here;
?>
```

You should simply pick one and stick with it throughout your program.

- **Frequent comments:** The more you will use comments throughout your code, the better off you will be. While it's not so important in smaller, simpler programs, when your programs become more and more complex, it will be hard for you to remember what you did, where you did it, and why you did it the way you did.
- **The use of line numbers:** Some text editors insert line numbers for you, but others do not. We recommend you use notpad++ or DW, but you should know that it is important to denote line numbers somehow in your code, if they are not provided for you, because PHP lets you know when your program generates errors, and it notifies you of the line number in which the error occurs. If you have to count the lines manually every time you encounter an error, you can imagine how time consuming and inefficient your debugging will be.
- **Caring about your code looks like:** It's important to follow good coding practices for three reasons:
  - ✓ **For efficiency:** The easier your code is to read and follow, the easier it will be to keep track of where you are with code, and the quicker it will be to pick up where you left off after a break.
  - ✓ **For debugging:** Knowing where your problem lies is a major debugging tool. If you have used comments, you can easily follow your own logic, and if you have line numbers and consistent formatting, you can easily scan your document to pinpoint a trouble area.
  - ✓ **For future expansions and modifications:** Utilizing comments in your code is especially important for future changes, as not all of us can remember the logic behind code that was written years or even just months ago. Also, if you

are working on code that involves a team, if every-one is utilizing the same coding styles, it will be much easier to make changes or additions to someone else's work down the road.

Creating your First Program

You can't get much simpler than this first program, but try it out to get a feel for what the results look like. The PHP function echo, seen in the material that follows, is one of the most commonly used PHP functions and one that, undoubtedly, you will become intimate with. It is used to send text (or variable values or a variety of other things) to the browser.

Try it out: Using echo

Try using echo to see what results you achieve:

1. Enter the following program in your favorite text editor (Notepad, WordPad, or whatever), and save it as firstprog.php.
   Make sure you save it in a "plain text" format to avoid parsing problems, and double-check to ensure that the file is not saved as firstprog.php.txt by default.

```
<HTML>
<HEAD>
<TITLE>My First PHP Program</TITLE>
</HEAD>
<BODY>
<?php
echo "<h1>I'm a lumberjack.</h1>";
?>
</BODY>
</HTML>
```

2. Open this program using your browser.

How it works

When a browser calls a PHP program, it first searches through the entire code line by line to locate all PHP sections (those encased in the appropriate tags) and it then process them one at a time. **To the server, all PHP code is treated as one line**, which is why your two lines of code were shown as one continuous line on the screen. After the PHP code has been parsed accordingly, the server goes back and gobbles up the remaining HTML and spits it out to the browser, PHP sections included.

Error messages and warnings

PHP tries to be helpful when problems arise. It provides error messages and warnings as follows:

- ✓ **Parse Error**: A Parse Error is a syntax error that PHP finds when it scans the script before executing it. A parse error is a fatal error, preventing the script from running at all. A parse error looks similar to the following:

  ```
  Parse error: parse error, error, in c:\test\test.php on line 6
  ```

  Often, you receive this error message because you've forgotten a semicolon, a parenthesis, or a curly brace. The error provides more information when possible. For instance, error might be unexpected T_ECHO, expecting ',' or ';'means that PHP found an echo statement where it was expecting a comma or a semicolon, which probably means you forgot the semicolon at the end of the previous line.
- ✓ **Error message**: You receive this message when PHP encounters a serious error during the execution of the program that prevents it from continuing to run. The message contains as much information as possible to help you identify the problem.

- ✓ **Warning message**: You receive this message when the program sees a problem but the problem is not serious enough to prevent the program from running. Warning messages do not mean that the program can't run; the program does continue to run. Rather, warning messages tell you that PHP believes that something is probably wrong. You should identify the source of the warning and then decide whether it needs to be fixed. It usually does.
- ✓ **Notice:** You receive a notice when PHP sees a condition that might be an error or might be perfectly okay. Notices, like warnings, do not cause the script to stop running. Notices are much less likely than warnings to indicate serious problems. Notices just tell you that you are doing something unusual and to take a second look at what you're doing to be sure that you really want to do it. One common reason why you might receive a notice is if you're echoing variables that don't exist. Here's an example of what you might see in that instance:

```
Notice: Undefined variable: age in testing.php on line 9
```

- ✓ **Strict**: Strict messages, added in **PHP 5**, warn about language that is poor coding practice or has been replaced by better code. All types of messages indicate the filename causing the problem and the line number where the problem was encountered. You can specify which types of error messages you want displayed in the Web page. In general, when you are developing a program, you want to see all messages, but when the program is published on your Web site, you do not want any messages to be displayed to the user. To change the error-message level for your Web site to show more or fewer messages, you must edit the php.ini file on your system. It contains a section that explains the error-message setting (error_reporting), error-message levels, and how to set them. Some possible settings are

```
error_reporting = E_ALL | E_STRICT
error_reporting = 0
error_reporting = E_ALL & ~ E_NOTICE
```

The first setting displays E_ALL, which is all errors, warnings, and notices except strict, and E_STRICT, which displays strict messages. The second setting displays no error messages. The third setting displays all error and warning messages, but not notices or stricts. After changing the error_reporting settings, save the edited php.ini file and restart your Web server. If you don't have access to php.ini, you can add a statement to a program that sets the error reporting level for that program only. Add the following statement at the beginning of the program:

```
error_reporting(errorSetting);
For example, to see all errors except stricts, use the following:
error_reporting(E_ALL);
```

Comments

PHP supports 'C', 'C++' and Unix shell-style (Perl style) comments. For example:

```
<?php
    echo 'This is a test'; // This is a one-line c++ style comment
    /* This is a multi line comment
       yet another line of comment */
    echo 'This is yet another test';
    echo 'One Final Test'; # This is a one-line shell-style comment
?>
```

The "one-line" comment styles only comment to the end of the line or the current block of PHP code, whichever comes first. This means that HTML code after // ... ?> or # ... ?> WILL be printed: ?> breaks out of PHP mode and returns to HTML mode, and // or #

cannot influence that. If the asp_tags configuration directive is enabled, it behaves the same with *// %>* and *# %>*. However, the *</script>* tag doesn't break out of PHP mode in a one-line comment.

```
<h1>This is an <?php # echo 'simple';?> example.</h1>
<p>The header above will say 'This is an  example'.</p>
```

'C' style comments end at the first */ encountered. Make sure you don't nest 'C' style comments. It is easy to make this mistake if you are trying to comment out a large block of code.

```
<?php
 /*
    echo 'This is a test'; /* This comment will cause a problem */
 */
?>
```

Using PHP Variables

Variables are containers used to hold information. A variable has a name, and information is stored in the variable. For instance, you might name a variable $age and store the number 12 in it. After information is stored in a variable, it can be used later in the program. One of the most common uses for variables is to hold the information that a user types into a form. Naming a variable when you're naming a variable, keep the following rules in mind:

- ✓ All variable names have a dollar sign ($) in front of them. This tells PHP that it is a variable name.
- ✓ Variable names can be any length.
- ✓ Variable names can include letters, numbers, and underscores only.
- ✓ Variable names must begin with a letter or an underscore. They cannot begin with a number.
- ✓ Uppercase and lowercase letters are not the same. For example, $firstname and $Firstname are not the same variable. If you store information in $firstname, for example, you can't access that information by using the variable name $firstName.
- ✓ A variable name should not contain spaces. If a variable name is more than one word, it should be separated with underscore ($my_string), or with capitalization ($myString)

PHP is a Loosely Typed Language

In PHP a variable does not need to be declared before being set. In the example above, you see that you do not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on how they are set. In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it. In PHP the variable is declared automatically when you use it.

Strings in PHP

String variables are used for values that contains character strings. In this tutorial we are going to look at some of the most common functions and operators used to manipulate strings in PHP. After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

Below, the PHP script assigns the string "Hello World" to a string variable called $txt:

```
<?php
$txt="Hello World";
echo $txt;
?>
```

The output of the code above will be: **Hello World**

Now, lets try to use some different functions and operators to manipulate our string.

The Concatenation Operator

There is only one string operator in PHP. The concatenation operator (.) is used to put two string values together. To concatenate two variables together, use the dot (.) operator:

```
<?php
$txt1="Hello World";
$txt2="1234";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string. Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string. Let's find the length of our string "Hello world!":

```
<?php
echo strlen("Hello world!");
?>
```

The output of the code above will be:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

Using the strpos() function

The strpos() function is used to search for a string or character within a string. If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

```
<?php
echo strpos("Hello world!","world");
?>
```

Heredoc

Another way to delimit strings is by using heredoc syntax ("<<<"). One should provide an identifier (followed by new line) after <<<, then the string, and then the same identifier to close the quotation.

The closing identifier *must* begin in the first column of the line. Also, the identifier used must follow the same naming rules as any other label in PHP: it must contain only alphanumeric characters and underscores, and must start with a non-digit character or underscore.

```
<?php
class foo {
    public $bar = <<<EOT
bar
EOT;
}
?>
```

Working with Numbers

PHP allows you to do arithmetic operations on numbers. You indicate arithmetic operations with two numbers and an arithmetic operator. For instance, one operator is the plus (+) sign, so you can indicate an arithmetic operation like this: 1 + 2

You can also perform arithmetic operations with variables that contain numbers, as follows:

```
$n1 = 1;
$n2 = 2;
$sum = $n1 + $n2;
```

Operator and it's descriptions:

+ = Add two numbers.

- = Subtract the second number from the first number.

***** = Multiply two numbers.

/ = Divide the first number by the second number.

% = Find the remainder when the first number is divided by the second number. This is called modulus. For instance, in $a = 13 % 4, $a is set to 1.

You can do several arithmetic operations at once. For instance, the following statement performs three operations: $result = 1 + 2 * 4 + 1;

The order in which the arithmetic is performed is important. You can get different results depending on which operation is performed first. PHP does multiplication and division first, followed by addition and subtraction. If other considerations are equal, PHP goes from left to right. Consequently, the preceding statement sets $result to 10, in the following order:

```
$result = 1 + 2 * 4 + 1    (first it does the multiplication)
$result = 1 + 8 + 1        (next it does the leftmost addition)
$result = 9 + 1            (next it does the remaining addition)
$result = 10
```

Using PHP Constants

PHP constants are similar to variables. Constants are given a name, and a value is stored in them. However, constants are constant; that is, they can't be changed by the program. After you set the value for a constant, it stays the same. If you used a constant for age and set it to 29, for example, it can't be changed. Wouldn't that be nice — 29 forever? Constants are used when a value is needed several places in the program and doesn't change during the program. The value is set in a constant at the start of the program. By using a constant throughout the program, instead of a variable, you make sure that the value won't get changed accidentally. By giving it a name, you know what the information is instantly. And by setting a constant once at the start of the program (instead of using the value throughout the program), you can change the value in one place if it needs changing instead of hunting for it in many places in the program to change it. For instance, you might set one constant that's the company name and another constant that's the company address and use them wherever needed. Then, if the company moves, you could just change the value in the company address at the start of the program instead of having to find every place in your program that echoed the company name to change it. Constants are set by using the define statement. The format is ***define("constantname","constantvalue");*** For instance, to set a constant with the company name, use the following statement:

```
define("COMPANY","ABC Pet Store");
```

Use the constant in your program wherever you need your company name:

```
echo COMPANY;
```

When you echo a constant, you can't enclose it in quotes. If you do, it will echo the constant name, instead of the value. You can echo it without anything, as shown in the preceding example, or enclosed with parentheses. You can use any name for a constant that you can use for a variable. Constant names are not preceded by a dollar sign ($). By convention, constants are given names that are all uppercase, so you can easily spot constants, but PHP itself doesn't care what you name a constant. You can store either a string or a number in it. The following statement is perfectly okay with PHP:

```
define ("AGE",29);
echo AGE;
```