PHP - Syntax

Before we talk about PHP's syntax, let us first define what syntax is referring to.

- **Syntax** - The rules that must be followed to write properly structured code.

PHP's syntax and semantics are similar to most other programming languages (C, Java, Perl) with the addition that all PHP code is contained with a tag, of sorts. All PHP code must be contained within the following...

PHP Code:

```
<?php
?>

or the shorthand PHP tag that requires shorthand support to be enabled
on your server...

<?
?>
```

If you are writing PHP scripts and plan on distributing them, we suggest that you use the standard form (which includes the ?php) rather than the shorthand form. This will ensure that your scripts will work, even when running on other servers with different settings.

How to Save Your PHP Pages

If you have PHP inserted into your HTML and want the web browser to interpret it correctly, then you must save the file with a *.php* extension, instead of the standard *.html* extension. So be sure to check that you are saving your files correctly. Instead of *index.html*, it should be *index.php* if there is PHP code in the file.

Example Simple HTML & PHP Page

Below is an example of one of the easiest PHP and HTML page that you can create and still follow web standards.

PHP and HTML Code:

```
<html>
<head>
<title>My First PHP Page</title>
</head>
<body>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Display:

```
Hello World!
```

If you save this file and place it on PHP enabled server and load it up in your web browser, then you should see "Hello World!" displayed. If not, please check that you followed our example correctly.

We used the PHP function *echo* to write "Hello World!" and we will be talking in greater depth about this PHP function and many others later on in this tutorial.

The Semicolon!

As you may or may not have noticed in the above example, there was a semicolon after the line of PHP code. The semicolon signifies the end of a PHP statement and should never be forgotten. For example, if we repeated our "Hello World!" code several times, then we would need to place a semicolon at the end of each statement.

PHP and HTML Code:

```
<html>
<head>
<title>My First PHP Page</title>
</head>
<body>
<?php
echo "Hello World! ";
echo "Hello World! ";
echo "Hello World! ";
echo "Hello World! ";
echo "Hello World! ";
?>
</body>
</html>
```

Display:

Hello World! Hello World! Hello World! Hello World! Hello World!

White Space

As with HTML, whitespace is ignored between PHP statements. This means it is OK to have one line of PHP code, then 20 lines of blank space before the next line of PHP code. You can also press tab to indent your code and the PHP interpreter will ignore those spaces as well.

PHP and HTML Code:

```
<html>
<head>
<title>My First PHP Page</title>
</head>
<body>
<?php
echo "Hello World!";
echo "Hello World!";
?>
</body>
</html>
```

Display:

```
Hello World!Hello World!
```

This is perfectly legal PHP code.

PHP - Variables

If you have never had any programming, Algebra, or scripting experience, then the concept of *variables* might be a new concept to you. A detailed explanation of variables is beyond the scope of this tutorial, but we've included a refresher crash course to guide you.

A variable is a means of storing a value, such as text string "Hello World!" or the integer value 4. A variable can then be reused throughout your code, instead of having to type out the actual value over and over again.

In PHP you define a variable with the following form:

- $variable_name = Value;

If you forget that dollar sign at the beginning, it will not work. This is a common mistake for new PHP programmers!

A Quick Variable Example

Say that we wanted to store the values that we talked about in the above paragraph. How would we go about doing this? We would first want to make a variable name and then set that equal to the value we want. See our example below for the correct way to do this.

PHP Code:

```
<?php
$hello = "Hello World!";
$a_number = 4;
$anotherNumber = 8;
?>
```

Note for programmers: PHP does not require variables to be declared before being initialized.

PHP Variable Naming Conventions

There are a few rules that you need to follow when choosing a name for your PHP variables.

- PHP variables must start with a letter or underscore "_".
- PHP variables may only be comprised of alpha-numeric characters and underscores. a-z, A-Z, 0-9, or _ .
- Variables with more than one word should be separated with underscores. $my_variable
- Variables with more than one word can also be distinguished with capitalization. $myVariable

PHP - Echo

As you saw in the previous lesson, the PHP function *echo* is a means of outputting text to the web browser. Throughout your PHP career you will be using the echo function more than any other. So let's give it a solid perusal!

Outputting a String

To output a string, like we have done in previous lessons, use the PHP echo function. You can place either a string variable or you can use quotes, like we do below, to create a string that the echo function will output.

PHP Code:

```
<?php
$myString = "Hello!";
echo $myString;
echo "<h5>I love using PHP!</h5>";
?>
```

Display:

Hello!

I love using PHP!

In the above example we output "Hello!" without a hitch. The text we are outputting is being sent to the user in the form of a web page, so it is important that we use proper HTML syntax!

In our second echo statement we use *echo* to write a valid Header 5 HTML statement. To do this we simply put the <h5> at the beginning of the string and closed it at the end of the string. Just because you're using PHP to make web pages does not mean you can forget about HTML syntax!

Careful When Echoing Quotes!

It is pretty cool that you can output HTML with PHP. **However, you must be careful when using HTML code or any other string that includes quotes!** The echo function uses quotes to define the beginning and end of the string, so you must use one of the following tactics if your string contains quotations:

- Don't use quotes inside your string
- Escape your quotes that are within the string with a slash. To escape a quote just place a slash directly before the quotation mark, i.e. \"
- Use single quotes (apostrophes) for quotes inside your string.

See our example below for the right and wrong use of the echo function:

PHP Code:

```
<?php
// This won't work because of the quotes around specialH5!
echo "<h5 class="specialH5">I love using PHP!</h5>";

// OK because we escaped the quotes!
echo "<h5 class=\"specialH5\">I love using PHP!</h5>";

// OK because we used an apostrophe '
echo "<h5 class='specialH5'>I love using PHP!</h5>";
?>
```

If you want to output a string that includes quotations, either use an apostrophe (') or *escape* the quotations by placing a slash in front of it (\"). The slash will tell PHP that you want the quotation to be used within the string and NOT to be used to end *echo's* string.

Echoing Variables

Echoing variables is very easy. The PHP developers put in some extra work to make the common task of echoing **all** variables nearly foolproof! No quotations are required, even if the variable does not hold a string. Below is the correct format for echoing a variable.

PHP Code:

```
<?php
$my_string = "Hello Bob.  My name is: ";
$my_number = 4;
$my_letter = a;
echo $my_string;
echo $my_number;
echo $my_letter;
?>
```

Display:

```
Hello Bob. My name is: 4a
```

Echoing Variables and Text Strings

You can also combine text strings and variables. By doing such a conjunction you save yourself from having to do a large number of echo statements. Variables and text strings are joined together with a period(.). The example below shows how to do such a combination.

PHP Code:

```
<?php
$my_string = "Hello Bob.  My name is: ";
$newline = "<br />";
echo $my_string."Bobettta".$newline;
echo "Hi, I'm Bob.  Who are you? ".$my_string.$newline;
echo "Hi, I'm Bob.  Who are you? ".$my_string."Bobetta";
?>
```

Display:

```
Hello Bob. My name is: Bobetta
Hi, I'm Bob. Who are you? Hello Bob. My name is:
Hi, I'm Bob. Who are you? Hello Bob. My name is: Bobetta
```

This combination can be done multiple times, as the example shows. This method of joining two or more strings together is called *concatenation* and we will talk more about this and other forms of string manipulation in our string lesson.

PHP - Strings

In the last lesson, PHP Echo, we used strings a bit, but didn't talk about them in depth. Throughout your PHP career you will be using strings a great deal, so it is important to have a basic understanding of PHP strings.

PHP - String Creation

Before you can use a string you have to create it! A string can be used directly in a function or it can be stored in a variable. Below we create the exact same string twice: first storing it into a variable and in the second case we place the string directly into a function.

PHP Code:

```
$my_string = "Tizag - Unlock your potential!";
echo "Tizag - Unlock your potential!";
echo $my_string;
```

In the above example the first string will be stored into the variable *$my_string*, while the second string will be used in the echo function and **not** be stored. Remember to save your strings into variables if you plan on using them more than once! Below is the output from our example code. They look identical just as we thought.

Display:

```
Tizag - Unlock your potential! Tizag - Unlock your potential!
```

PHP - String Creation Single Quotes

Thus far we have created strings using double-quotes, but it is just as correct to create a string using single-quotes, otherwise known as apostrophes.

PHP Code:

```
$my_string = 'Tizag - Unlock your potential!';
echo 'Tizag - Unlock your potential!';
echo $my_string;
```

If you want to use a single-quote within the string you have to *escape* the single-quote with a backslash \ . Like this: \' !

PHP Code:

```
echo 'Tizag - It\'s Neat!';
```

PHP - String Creation Double-Quotes

We have used double-quotes and will continue to use them as the primary method for forming strings. Double-quotes allow for many special escaped characters to be used that you cannot do with a single-quote string. Once again, a backslash is used to escape a character.

PHP Code:

```
$newline = "A newline is \n";

$return = "A carriage return is \r";

$tab = "A tab is \t";

$dollar = "A dollar sign is \$";

$doublequote = "A double-quote is \"";
```

Note: If you try to escape a character that doesn't need to be, such as an apostrophe, then the backslash will show up when you output the string.

These escaped characters are not very useful for outputting to a web page because HTML ignore extra white space. A tab, newline, and carriage return are all examples of extra (ignorable) white space. However, when writing to a file that may be read by human eyes these escaped characters are a valuable tool!

PHP - String Creation Heredoc

The two methods above are the traditional way to create strings in most programming languages. PHP introduces a more robust string creation tool called *heredoc* that lets the programmer create multi-line strings without using quotations. However, creating a string using heredoc is more difficult and can lead to problems if you do not properly code your string! Here's how to do it:

PHP Code:

```
$my_string = <<<TEST

Tizag.com

Webmaster Tutorials

Unlock your potential!

TEST;


echo $my_string;
```

There are a few **very** important things to remember when using heredoc.

- Use <<< and some identifier that you choose to begin the heredoc. In this example we chose *TEST* as our identifier.
- Repeat the identifier followed by a semicolon to end the heredoc string creation. In this example that was *TEST;*
- The closing sequence *TEST;* must occur on a line by itself and **cannot** be indented!

Another thing to note is that when you output this multi-line string to a web page, it will not span multiple lines because we did not have any
 tags contained inside our string! Here is the output made from the code above.

Display:

```
Tizag.com Webmaster Tutorials Unlock your potential!
```

Once again, take great care in following the heredoc creation guidelines to avoid any headaches.

PHP - Operators

In all programming languages, operators are used to manipulate or perform operations on variables and values. You have already seen the string concatenation operator "." in the Echo Lesson and the assignment operator "=" in pretty much every PHP example so far.

There are many operators used in PHP, so we have separated them into the following categories to make it easier to learn them all.

- Assignment Operators
- Arithmetic Operators
- Comparison Operators
- String Operators
- Combination Arithmetic & Assignment Operators

Assignment Operators

Assignment operators are used to set a variable equal to a value or set a variable to another variable's value. Such an assignment of value is done with the "=", or equal character. Example:

- $my_var = 4;
- $another_var = $my_var

Now both $my_var and $another_var contain the value 4. Assignments can also be used in conjunction with arithmetic operators.

Arithmetic Operators

Operator	English	Example
+	Addition	2 + 4
-	Subtraction	6 - 2
*	Multiplication	5 * 3
/	Division	15 / 3
%	Modulus	43 % 10

PHP Code:

```
$addition = 2 + 4;
$subtraction = 6 - 2;
$multiplication = 5 * 3;
$division = 15 / 3;
$modulus = 5 % 2;
echo "Perform addition: 2 + 4 = ".$addition."<br />";
echo "Perform subtraction: 6 - 2 = ".$subtraction."<br />";
echo "Perform multiplication:  5 * 3 = ".$multiplication."<br />";
echo "Perform division: 15 / 3 = ".$division."<br />";
echo "Perform modulus: 5 % 2 = " . $modulus
        . ". Modulus is the remainder after the division operation has been performed.
        In this case it was 5 / 2, which has a remainder of 1.";
```

Display:

```
Perform addition: 2 + 4 = 6
Perform subtraction: 6 - 2 = 4
Perform multiplication: 5 * 3 = 15
Perform division: 15 / 3 = 5
Perform modulus: 5 % 2 = 1. Modulus is the remainder after the division operation has been
performed. In this case it was 5 / 2, which has a remainder of 1.
```

Comparison Operators

Comparisons are used to check the relationship between variables and/or values. If you would like to see a simple example of a comparison operator in action, check out our If Statement Lesson. Comparison operators are used inside conditional statements and evaluate to either *true* or *false*. Here are the most important comparison operators of PHP.

Assume: $x = 4 and $y = 5;

Operator	English	Example	Result
==	Equal To	$x == $y	false
!=	Not Equal To	$x != $y	true
<	Less Than	$x < $y	true
>	Greater Than	$x > $y	false
<=	Less Than or Equal To	$x <= $y	true
>=	Greater Than or Equal To	$x >= $y	false

String Operators

As we have already seen in the Echo Lesson, the period "." is used to add two strings together, or more technically, the period is the concatenation operator for strings.

PHP Code:

```
$a_string = "Hello";
$another_string = " Billy";
$new_string = $a_string . $another_string;
echo $new_string . "!";
```

Display:

```
Hello Billy!
```

Combination Arithmetic & Assignment Operators

In programming it is a very common task to have to increment a variable by some fixed amount. The most common example of this is a counter. Say you want to increment a counter by 1, you would have:

- $counter = $counter + 1;

However, there is a shorthand for doing this.

- $counter += 1;

This combination assignment/arithmetic operator would accomplish the same task. The downside to this combination operator is that it reduces code readability to those programmers who are not used to such an operator. Here are some examples of other common shorthand operators. In general, "+=" and "-=" are the most widely used combination operators.

Operator	English	Example	Equivalent Operation
+=	Plus Equals	$x += 2;	$x = $x + 2;
-=	Minus Equals	$x -= 4;	$x = $x - 4;
*=	Multiply Equals	$x *= 3;	$x = $x * 3;
/=	Divide Equals	$x /= 2;	$x = $x / 2;
%=	Modulo Equals	$x %= 5;	$x = $x % 5;
.=	Concatenate Equals	$my_str.="hello";	$my_str = $my_str . "hello";

Pre/Post-Increment & Pre/Post-Decrement

This may seem a bit absurd, but there is even a shorter shorthand for the common task of adding 1 or subtracting 1 from a variable. To add one to a variable or "increment" use the "++" operator:

- $x++; Which is equivalent to $x += 1; or $x = $x + 1;

To subtract 1 from a variable, or "decrement" use the "--" operator:

- $x--; Which is equivalent to $x -= 1; or $x = $x - 1;

In addition to this "shorterhand" technique, you can specify whether you want the increment to before the line of code is being executed or after the line has executed. Our PHP code below will display the difference.

PHP Code:

```
$x = 4;
echo "The value of x with post-plusplus = " . $x++;
echo "<br /> The value of x after the post-plusplus is " . $x;
$x = 4;
echo "<br />The value of x with with pre-plusplus = " . ++$x;
echo "<br /> The value of x after the pre-plusplus is " . $x;
```

Display:

```
The value of x with post-plusplus = 4
The value of x after the post-plusplus is = 5
The value of x with with pre-plusplus = 5
The value of x after the pre-plusplus is = 5
```

As you can see the value of $x++ is not reflected in the echoed text because the variable is not incremented until after the line of code is executed. However, with the pre-increment "++$x" the variable does reflect the addition immediately.

Using Comments in PHP

Comments in PHP are similar to comments that are used in HTML. The PHP comment syntax always begins with a special character sequence and all text that appears between the start of the comment and the end will be ignored by the browser.

In HTML a comment's main purpose is to serve as a note to you, the web developer or to others who may view your website's source code. However, PHP's comments are different in that they will not be displayed to your visitors. The only way to view PHP comments is to open the PHP file for editing. This makes PHP comments only useful to PHP programmers.

In case you forgot what an HTML comment looked like, see our example below.

HTML Code:

```
<!--- This is an HTML Comment -->
```

PHP Comment Syntax: Single Line Comment

While there is only one type of comment in HTML, PHP has two types. The first type we will discuss is the single line comment. The single line comment tells the interpreter to ignore everything that occurs on that line to the right of the comment. To do a single line comment type "//" and all text to the right will be ignored by PHP interpreter.

PHP Code:

```
<?php
echo "Hello World!"; // This will print out Hello World!
echo "<br />Psst...You can't see my PHP comments!"; // echo "nothing";
// echo "My name is Humperdinkle!";
?>
```

Display:

```
Hello World!
Psst...You can't see my PHP comments!
```

Notice that a couple of our echo statements were not evaluated because we commented them out with the single line comment. This type of line commenting is often used for quick notes about complex and confusing code or to temporarily remove a line of PHP code.

PHP Comment Syntax: Multiple Line Comment

Similiar to the HTML comment, the multi-line PHP comment can be used to comment out large blocks of code or writing multiple line comments. The multiple line PHP comment begins with " /* " and ends with " */ ".

PHP Code:

```
<?php
/* This Echo statement will print out my message to the
the place in which I reside on.  In other words, the World. */
echo "Hello World!";
/* echo "My name is Humperdinkle!";
echo "No way! My name is Uber PHP Programmer!";
*/
?>
```

Display:

```
Hello World!
```

Good Commenting Practices

One of the best commenting practices that I can recommend to new PHP programmers is....USE THEM!! So many people write complex PHP code and are either too lazy to write good comments or believe the commenting is not needed. However, do you really believe that you will remember exactly what you were thinking when looking at this code a year or more down the road?

Let the comments permeate your code and you will be a happier PHPer in the future. Use single line comments for quick notes about a tricky part in your code and use multiple line comments when you need to describe something in greater depth than a simple note.

The Include Function

Without understanding much about the details of PHP, you can save yourself a great deal of time with the use of the PHP *include* function. The *include* function takes a file name and simply inserts that file's contents into the script that calls used the *include* function.

Why is this a cool thing? Well, first of all, this means that you can type up a common header or menu file that you want all your web pages to include. When you add a new page to your site, instead of having to update the links on several web pages, you can simply change the Menu file.

An Include Example

Say we wanted to create a common menu file that all our pages will use. A common practice for naming files that are to be included is to use the ".php" extension. Since we want to create a common menu let's save it as "menu.php".

menu.php Code:

```
<html>
<body>
<a href="http://www.example.com/index.php">Home</a> -
<a href="http://www.example.com/about.php">About Us</a> -
<a href="http://www.example.com/links.php">Links</a> -
<a href="http://www.example.com/contact.php">Contact Us</a> <br />
```

Save the above file as "menu.php". Now create a new file, "index.php" in the same directory as "menu.php". Here we will take advantage of the *include* function to add our common menu.

index.php Code:

```
<?php include("menu.php"); ?>
<p>This is my home page that uses a common menu to save me time when I add
new pages to my website!</p>
</body>
</html>
```

Display:

Home - About Us - Links - Contact Us

This is my home page that uses a common menu to save me time when I add new pages to my website!

And we would do the same thing for "about.php", "links.php", and "contact.php". Just think how terrible it would be if you had 15 or more pages with a common menu and you decided to add another web page to that site. You would have to go in an manually edit every single file to add this new page, but with include files you simply have to change "menu.php" and all your problems are solved. Avoid such troublesome occasions with a simple include file.

What do Visitors See?

If we were to use the include function to include a common menu on each of our web pages, what would the visitor see if they viewed the source of "index.php"? Well, because the include function is pretty much the same as copying and pasting, the visitors would see:

View Source of index.php to a Visitor:

```
<html>
<body>
<a href="index.php">Home</a> -
<a href="about.php">About Us</a> -
<a href="links.php">Links</a> -
<a href="contact.php">Contact Us</a> <br />
<p>This is my home page that uses a common menu to save me time when I add
new pages to my website!</p>
</body>
</html>
```

The visitor would actually see all the HTML code as one long line of HTML code, because we have not inserted any new line characters. We did some formatting above to make it easier to read. We will be discussing new line characters later.

Include Recap

The include command simply takes all the text that exists in the specified file and copies it into the file that uses the include function. Include is quite useful when you want to include the same PHP, HTML, or text segment on multiple pages of a website. The include function is used widely by PHP web developers.

The next lesson will talk about a slight variation of the include function: the *require* function. It is often best to use the require function instead of the include function in your PHP Code. Read the next lesson to find out why!

PHP Require Function

Just like the previous lesson, the require function is used to include a file into your PHP code. However there is one huge difference between the two functions, though it might not seem that big of a deal.

Require vs Include

When you include a file with the *include* function and PHP cannot find it you will see an error message like the following:

PHP Code:

```
<?php
include("noFileExistsHere.php");
echo "Hello World!";
?>
```

Display:

Warning: main(noFileExistsHere.php): failed to open stream: No such file or directory in **/home/websiteName/FolderName/tizagScript.php** on line **2** Warning: main(): Failed opening 'noFileExistsHere.php' for inclusion (include_path='.:/usr/lib/php:/usr/local/lib/php') in **/home/websiteName/FolderName/tizagScript.php** on line 2

Hello World!

Notice that our echo statement is still executed, this is because a Warning does not prevent our PHP script from running. On the other hand, if we did the same example but used the require statement we would get something like the following example.

PHP Code:

```
<?php
require("noFileExistsHere.php");
echo "Hello World!";
?>
```

Display:

Warning: main(noFileExistsHere.php): failed to open stream: No such file or directory in **/home/websiteName/FolderName/tizagScript.php** on line **2**
Fatal error: main(): Failed opening required 'noFileExistsHere.php' (include_path='.:/usr/lib/php:/usr/local/lib/php') in **/home/websiteName/FolderName/tizagScript.php** on line 2

The echo statement was not executed because our script execution died after the *require* function returned a fatal error! We recommend that you use require instead of include because your scripts should not be executing if necessary files are missing or misnamed.

The If Statement

The PHP *if statement* is very similar to other programming languages use of the *if statement*, but for those who are not familiar with it, picture the following:

Think about the decisions you make before you go to sleep. **If** you have something to do the next day, say go to work, school, or an appointment, **then** you will set your alarm clock to wake you up. **Otherwise**, you will sleep in as long as you like!

This simple kind of if/then statement is very common in every day life and also appears in programming quite often. Whenever you want to make a decision given that something is true (you have something to do tomorrow) and be sure that you take the appropriate action, you are using an if/then relationship.

The PHP If Statement

The if statement is necessary for most programming, thus it is important in PHP. Imagine that on January 1st you want to print out "Happy New Year!" at the top of your personal web page. With the use of PHP *if statements* you could have this process automated, months in advance, occuring every year on January 1st.

This idea of planning for future events is something you would never have had the opportunity of doing if you had just stuck with HTML.

If Statement Example

The "Happy New Year" example would be a little difficult for you to do right now, so let us instead start off with the basics of the if statement. The PHP if statement tests to see if a value is true, and if it is a segment of code will be executed. See the example below for the form of a PHP if statement.

PHP Code:

```
$my_name = "someguy";

if ( $my_name == "someguy" ) {
   echo "Your name is someguy!<br />";
}
echo "Welcome to my homepage!";
```

Display:

```
Your name is someguy!
Welcome to my homepage!
```

Did you get that we were comparing the variable $my_name with "someguy" to see if they were equal? In PHP you use the double equal sign (==) to compare values. Additionally, notice that because the if statement turned out to be true, the code segment was executed, printing out "Your name is someguy!". Let's go a bit more in-depth into this example to iron out the details.

- We first set the variable $my_name equal to "someguy".
- We next used a PHP if statement to check if the value contained in the variable $my_name was equal to "someguy"
- The comparison between $my_name and "someguy" was done with a double equal sign "==", not a single equals"="! A single equals is for assigning a value to a variable, while a double equals is for checking if things are equal.
- Translated into english the PHP statement ($my_name == "someguy") is ($my_name is equal to "someguy").
- $my_name is indeed equal to "someguy" so the echo statement is executed.

A False If Statement

Let us now see what happens when a PHP if statement is not true, in other words, false. Say that we changed the above example to:

PHP Code:

```
$my_name = "anotherguy";

if ( $my_name == "someguy" ) {
        echo "Your name is someguy!<br />";
}
echo "Welcome to my homepage!";
```

Display:

```
Welcome to my homepage!
```

Here the variable contained the value "anotherguy", which is not equal to "someguy". The if statement evaluated to *false*, so the code segment of the if statement was not executed. When used properly, the *if statement* is a powerful tool to have in your programming arsenal!

If/Else Conditional Statment

Has someone ever told you, "if you work hard, then you will succeed"? And what happens if you do not work hard? Well, you fail! This is an example of an if/else conditional statement.

- If you work hard then you will succeed.
- Else, if you do not work hard, then you will fail.

How does this translate into something useful for PHP developers? Well consider this:
Someone comes to your website and you want to ask this visitor her name if it is her first time coming to your site. With an if statement this is easy. Simply have a conditional statement to check, "are you visiting for the first time". If the condition is true, then take them to the "Insert Your Name" page, else let her view the website as normal because you have already asked her for her name in the past.

If/Else an Example

Using these conditional statements can add a new layers of "cool" to your website. Here's the basic form of an if/else statement in PHP.

PHP Code:

```
$number_three = 3;

if ( $number_three == 3 ) {
        echo "The if statement evaluated to true";
} else {
        echo "The if statement evaluated to false";
}
```

Display:

```
The if statement evaluated to true
```

This is a lot to digest in one sitting, so let us step through the code, line by line.

- We first made a PHP variable called $number_three and set it equal to 3.
- In this example we compared a variable to an integer value. To do such a comparison we use "**==**", which in English means "Is Equal To".
- $number_three is indeed Equal To 3 and so this statement will evaluate to true.
- All code that is contained between the opening curly brace "{" that follows the if statement and the closing curly brace "}" will be executed when the if statement is true.
- The code contained within the else segment will **not** used.

Execute Else Code with False

On the other hand, if the *if statement* was false, then the code contained in the else segment would have been executed. Note that the code within the *if* and *else* cannot both be executed, as the if statement cannot evaluate to both true and false at one time! Here is what would happen if we changed to $number_three to anything besides the number 3.

PHP Code:

```
$number_three = 421;

if ( $number_three == 3 ) {
        echo "The if statement evaluated to true";
} else {
        echo "The if statement evaluated to false";
}
```

Display:

```
The if statement evaluated to false
```

The variable was set to 421, which is not equal to 3 and the if statement was false. As you can see, the code segment contained within the else was used in this case.

PHP - Elseif

An if/else statement is great if you only need to check for one condition. However, what would you do if you wanted to check if your *$employee* variable was the company owner Bob, the Vice President Ms. Tanner, or a regular employee? To check for these different conditions you would need the *elseif* statement.

PHP - Elseif What is it?

An *if* statement is made up of the keyword "if" and a conditional statement (i.e. $name == "Ted"). Just like an if statement, an *elseif* statement also contains a conditional statement, but it must be preceded by an *if* statement. You cannot have an *elseif* statement without first having an *if* statement.

When PHP evaluates your If...elseif...else statement it will first see if the If statement is true. If that tests comes out false it will then check the first elseif statement. If that is false it will either check the next elseif statement, or if there are no more elseif statements, it will evaluate the else segment, if one exists (I don't think I've ever used the word "if" so much in my entire life!). Let's take a look at a real world example.

PHP - Using Elseif with If...Else

Let's start out with the base case. Imagine we have a simpler version of the problem described above. We simply want to find out if the employee is the Vice President Ms. Tanner. We only need an *if else* statement for this part of the example.

PHP Code:

```
$employee = "Bob";
if($employee == "Ms. Tanner"){
        echo "Hello Ma'am";
} else {
        echo "Morning";
}
```

Now, if we wanted to also check to see if the big boss Bob was the employee we need to insert an *elseif* clause.

PHP Code:

```
$employee = "Bob";
if($employee == "Ms. Tanner"){
        echo "Hello Ma'am";
} elseif($employee == "Bob"){
        echo "Good Morning Sir!";
}else {
        echo "Morning";
}
```

Display:

```
Good Morning Sir!
```

PHP first checked to see if *$employee* was equal to "Ms. Tanner", which evaluated to false. Next, PHP checked the first elseif statement. *$employee* did in fact equal "Bob" so the phrase "Good Morning Sir!" was printed out. If we wanted to check for more employee names we could insert more elseif statements!

Remember that an elseif statement cannot be used unless it is preceded by an if statement!

PHP Switch Statement

In the previous lessons we covered the various elements that make up an If Statement in PHP. However, there are times when an if statement is not the most efficient way to check for certain conditions.

For example we might have a variable that stores travel destinations and you want to pack according to this destination variable. In this example you might have 20 different locations that you would have to check with a nasty long block of If/ElseIf/ElseIf/ElseIf/... statements. This doesn't sound like much fun to code, let's see if we can do something different.

PHP Switch Statement: Speedy Checking

With the use of the *switch* statement you can check for all these conditions at once, and the great thing is that it is actually more efficient programming to do this. A true win-win situation!

The way the Switch statement works is it takes a single variable as input and then checks it against all the different *cases* you set up for that *switch* statement. Instead of having to check that variable one at a time, as it goes through a bunch of If Statements, the Switch statement only has to check one time.

PHP Switch Statement Example

In our example the single variable will be *$destination* and the cases will be: Las Vegas, Amsterdam, Egypt, Tokyo, and the Caribbean Islands.

PHP Code:

```
$destination = "Tokyo";
echo "Traveling to $destination<br />";
switch ($destination){
        case "Las Vegas":
                echo "Bring an extra $500";
                break;
        case "Amsterdam":
                echo "Bring an open mind";
                break;
        case "Egypt":
                echo "Bring 15 bottles of SPF 50 Sunscreen";
                break;
        case "Tokyo":
                echo "Bring lots of money";
                break;
        case "Caribbean Islands":
                echo "Bring a swimsuit";
                break;
}
```

Display:

```
Traveling to Tokyo
Bring lots of money
```

The value of $destination was Tokyo, so when PHP performed the *switch* operating on $destination in immediately did a search for a *case* with the value of "Tokyo". It found it and proceeded to execute the code that existed within that segment.

You might have noticed how each case contains a *break;* at the end of its code area. This *break* prevents the other cases from being executed. If the above example did not have any break statements then all the cases that follow Tokyo would have been executed as well. Use this knowledge to enhance the power of your switch statements!

The form of the switch statement is rather unique, so spend some time reviewing it before moving on. Note: Beginning programmers should always include the *break;* to avoid any unnecessary confusion.

PHP Switch Statement: Default Case

You may have noticed the lack of a place for code when the variable doesn't match our condition. The *if* statement has the *else* clause and the *switch* statement has the *default* case.

It's usually a good idea to always include the *default* case in all your switch statements. Below is a variation of our example that will result in none of the cases being used causing our switch statement to fall back and use the default case. **Note**: there is no *case* before *default*.

PHP Code:

```
$destination = "New York";
echo "Traveling to $destination<br />";
switch ($destination){
        case "Las Vegas":
                echo "Bring an extra $500";
                break;
        case "Amsterdam":
                echo "Bring an open mind";
                break;
        case "Egypt":
                echo "Bring 15 bottles of SPF 50 Sunscreen";
                break;
        case "Tokyo":
                echo "Bring lots of money";
                break;
        case "Caribbean Islands":
                echo "Bring a swimsuit";
                break;
        default:
                echo "Bring lots of underwear!";
                break;
}
```

Display:

```
Traveling to New York
Bring lots of underwear!
```

Using PHP With HTML Forms

It is time to apply the knowledge you have obtained thus far and put it to real use. A very common application of PHP is to have an HTML form gather information from a website's visitor and then use PHP to do process that information. In this lesson we will simulate a small business's website that is implementing a very simple order form.

Imagine we are an art supply store that sells brushes, paint, and erasers. To gather order information from our prospective customers we will have to make a page with an HTML form to gather the customer's order.

Note: This is an oversimplified example to educate you how to use PHP to process HTML form information. This example is not intended nor advised to be used on a real business website.

Creating the HTML Form

If you need a refresher on how to properly make an HTML form, check out the HTML Form Lesson before continuing on.

We first create an HTML form that will let our customer choose what they would like to purchase. This file should be saved as "order.html"

.

order.html Code:

```
<html><body>
<h4>Tizag Art Supply Order Form</h4>
<form>
<select>
<option>Paint</option>
<option>Brushes</option>
<option>Erasers</option>
</select>
Quantity: <input type="text" />
<input type="submit" />
</form>
</body></html>
```

Display:

Tizag Art Supply Order Form

Paint Quantity: Submit Query

Remember to review HTML Forms if you do not understand any of the above HTML code. Next we must alter our HTML form to specify the PHP page we wish to send this information to. Also, we set the method to "post".

order.html Code:

```
<html><body>
<h4>Tizag Art Supply Order Form</h4>
<form action="process.php" method="post">
<select name="item">
<option>Paint</option>
<option>Brushes</option>
<option>Erasers</option>
</select>
Quantity: <input name="quantity" type="text" />
<input type="submit" />
</form>
</body></html>
```

Now that our "order.html" is complete, let us continue on and create the "process.php" file which will process the HTML form information.

PHP Form Processor

We want to get the "item" and "quantity" *inputs* that we have specified in our HTML form. Using an associate array (this term is explained in the array lesson), we can get this information from the $_POST associative array.

The proper way to get this information would be to create two new variables, $item and $quantity and set them equal to the values that have been "posted". The name of this file is "process.php".

process.php Code:

```
<html><body>
<?php
$quantity = $_POST['quantity'];
$item = $_POST['item'];

echo "You ordered ". $quantity . " " . $item . ".<br />";
echo "Thank you for ordering from Tizag Art Supplies!";

?>
</body></html>
```

As you probably noticed, the *name* in $_POST['*name*'] corresponds to the name that we specified in our HTML form.

Now try uploading the "order.html" and "process.php" files to a PHP enabled server and test them out. If someone selected the item brushes and specified a quantity of 6, then the following would be displayed on "process.php":

process.php Code:

```
You ordered 6 brushes.
Thank you for ordering from Tizag Art Supplies!
```

PHP & HTML Form Review

A lot of things were going on in this example. Let us step through it to be sure you understand what was going on.

1. We first created an HTML form "order.html" that had two input fields specified, "item" and "quantity".
2. We added two attributes to the form tag to point to "process.php" and set the method to "post".
3. We had "process.php" get the information that was posted by setting new variables equal to the values in the $_POST associative array.
4. We used the PHP echo function to output the customers order.

Remember, this lesson is only to teach you how to use PHP to get information from HTML forms. The example on this page should not be used for a real business.

PHP - Functions

A function is just a name we give to a block of code that can be executed whenever we need it. This might not seem like that big of an idea, but believe me, when you understand and use functions you will be able to save a ton of time and write code that is much more readable!

For example, you might have a company motto that you have to display at least once on every webpage. If you don't, then you get fired! Well, being the savvy PHP programmer you are, you think to yourself, "this sounds like a situation where I might need functions."

Tip: Although functions are often thought of as an advanced topic for beginning programmers to learn, if you take it slow and stick with it, functions can be just minor speedbump in your programming career. So don't give up if you functions confuse you at first!

Creating Your First PHP Function

When you create a function, you first need to give it a name, like *myCompanyMotto*. It's with this function name that you will be able to call upon your function, so make it easy to type and understand.

The actual syntax for creating a function is pretty self-explanatory, but you can be the judge of that. First, you must tell PHP that you want to create a function. You do this by typing the keyword *function* followed by your function name and some other stuff (which we'll talk about later).

Here is how you would make a function called *myCompanyMotto*. **Note**: We still have to fill in the code for *myCompanyMotto*.

PHP Code:

```
<?php
function myCompanyMotto(){
}
?>
```

Note: Your function name can start with a letter or underscore "_", but **not** a number!

With a properly formatted function in place, we can now fill in the code that we want our function to execute. Do you see the curly braces in the above example "{ }"? These braces define where our function's code goes. The opening curly brace "{" tells php that the function's code is starting and a closing curly brace "}" tells PHP that our function is done!

We want our function to print out the company motto each time it's called, so that sounds like it's a job for the *echo* function!

PHP Code:

```
<?php
function myCompanyMotto(){
   echo "We deliver quantity, not quality!<br />";
}
?>
```

That's it! You have written your first PHP function from scratch! Notice that the code that appears within a function is just the same as any other PHP code.

Using Your PHP Function

Now that you have completed coding your PHP function, it's time to put it through a test run. Below is a simple PHP script. Let's do two things: add the function code to it and use the function twice.

PHP Code:

```
<?php
echo "Welcome to Tizag.com <br />";
echo "Well, thanks for stopping by! <br />";
echo "and remember... <br />";
?>
```

PHP Code with Function:

```
<?php
function myCompanyMotto(){
   echo "We deliver quantity, not quality!<br />";
}
echo "Welcome to Tizag.com <br />";
myCompanyMotto();
echo "Well, thanks for stopping by! <br />";
echo "and remember... <br />";
myCompanyMotto();
?>
```

Display:

```
Welcome to Tizag.com
We deliver quantity, not quality!
Well, thanks for stopping by!
and remember...
We deliver quantity, not quality!
```

Although this was a simple example, it's important to understand that there is a lot going on and there are a lot of areas to make errors. When you are creating a function, follow these simple guidelines:

- Always start your function with the keyword **function**
- Remember that your function's code must be between the "{" and the "}"
- When you are using your function, be sure you spell the function name correctly
- Don't give up!

PHP Functions - Parameters

Another useful thing about functions is that you can send them information that the function can then use. Our first function *myCompanyMotto* isn't all that useful because all it does, and ever will do, is print out a single, unchanging string.

However, if we were to use parameters, then we would be able to add some extra functionality! A parameter appears with the parentheses "( )" and looks just like a normal PHP variable. Let's create a new function that creates a custom greeting based off of a person's name.

Our parameter will be the person's name and our function will concatenate this name onto a greeting string. Here's what the code would look like.

PHP Code with Function:

```
<?php
function myGreeting($firstName){
   echo "Hello there ". $firstName . "!<br />";
}
?>
```

When we use our *myGreeting* function we have to send it a string containing someone's name, otherwise it will break. When you add parameters, you also add more responsibility to you, the programmer! Let's call our new function a few times with some common first names.

PHP Code:

```
<?php
function myGreeting($firstName){
   echo "Hello there ". $firstName . "!<br />";
}
myGreeting("Jack");
myGreeting("Ahmed");
myGreeting("Julie");
myGreeting("Charles");
?>
```

Display:

```
Hello there Jack!
Hello there Ahmed!
Hello there Julie!
Hello there Charles!
```

It is also possible to have multiple parameters in a function. To separate multiple parameters PHP uses a comma ",". Let's modify our function to also include last names.

PHP Code:

```
<?php
function myGreeting($firstName, $lastName){
   echo "Hello there ". $firstName ." ". $lastName ."!<br />";
}
myGreeting("Jack", "Black");
myGreeting("Ahmed", "Zewail");
myGreeting("Julie", "Roberts");
myGreeting("Charles", "Schwab");
?>
```

Display:

```
Hello there Jack Black!
Hello there Ahmed Zewail!
Hello there Julie Roberts!
Hello there Charles Schwab!
```

PHP Functions - Returning Values

Besides being able to pass functions information, you can also have them return a value. However, a function can only return one thing, although that thing can be any integer, float, array, string, etc. that you choose!

How does it return a value though? Well, when the function is used and finishes executing, it sort of changes from being a function name into being a value. To capture this value you can set a variable equal to the function. Something like:

- $myVar = somefunction();

Let's demonstrate this returning of a value by using a simple function that returns the sum of two integers.

PHP Code:

```
<?php
function mySum($numX, $numY){
   $total = $numX + $numY;
   return $total;
}
$myNumber = 0;
echo "Before the function, myNumber = ". $myNumber ."<br />";
$myNumber = mySum(3, 4); // Store the result of mySum in $myNumber
echo "After the function, myNumber = " . $myNumber ."<br />";
?>
```

Display:

```
Before the function, myNumber = 0
After the function, myNumber = 7
```

When we first print out the value of $myNumber it is still set to the original value of 0. However, when we set $myNumber equal to the function *mySum*, $myNumber is set equal to mySum's result. In this case, the result was 3 + 4 = 7, which was successfully stored into $myNumber and displayed in the second echo statement!

PHP Functions - Practice Makes Perfect

If you are new to programming, then this lesson might or might not seem like overkill. If you are having a hard time understanding lessons, the best piece of advice would be to do your best the first time, then be sure to come back tomorrow and next week and see if it makes anymore sense. Chances are, after going through this tutorial more than once, with breaks in between, this topic will be mastered.

PHP - Arrays

An array is a data structure that stores one or more values in a single value. For experienced programmers it is important to note that PHP's arrays are actually maps (each key is mapped to a value).

PHP - A Numerically Indexed Array

If this is your first time seeing an array, then you may not quite understand the concept of an array. Imagine that you own a business and you want to store the names of all your employees in a PHP variable. How would you go about this?

It wouldn't make much sense to have to store each name in its own variable. Instead, it would be nice to store all the employee names inside of a single variable. This can be done, and we show you how below.

PHP Code:

```
$employee_array[0] = "Bob";
$employee_array[1] = "Sally";
$employee_array[2] = "Charlie";
$employee_array[3] = "Clare";
```

In the above example we made use of the *key / value* structure of an array. The *keys* were the numbers we specified in the array and the *values* were the names of the employees. Each key of an array represents a value that we can manipulate and reference. The general form for setting the key of an array equal to a value is:

- $array[*key*] = *value*;

If we wanted to reference the values that we stored into our array, the following PHP code would get the job done.

PHP Code:

```
echo "Two of my employees are "
. $employee_array[0] . " & " . $employee_array[1];
echo "<br />Two more employees of mine are "
. $employee_array[2] . " & " . $employee_array[3];
```

Display:

```
Two of my employees are Bob & Sally
Two more employees of mine are Charlie & Clare
```

PHP arrays are quite useful when used in conjunction with loops, which we will talk about in a later lesson. Above we showed an example of an array that made use of integers for the *keys* (a numerically indexed array). However, you can also specify a string as the *key*, which is referred to as an associative array.

PHP - Associative Arrays

In an associative array a key is associated with a value. If you wanted to store the salaries of your employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the *keys* in our associative array, and the *value* would be their respective salary.

PHP Code:

```
$salaries["Bob"] = 2000;
$salaries["Sally"] = 4000;
$salaries["Charlie"] = 600;
$salaries["Clare"] = 0;

echo "Bob is being paid - $" . $salaries["Bob"] . "<br />";
echo "Sally is being paid - $" . $salaries["Sally"] . "<br />";
echo "Charlie is being paid - $" . $salaries["Charlie"] . "<br />";
echo "Clare is being paid - $" . $salaries["Clare"];
```

Display:

```
Bob is being paid - $2000
Sally is being paid - $4000
Charlie is being paid - $600
Clare is being paid - $0
```

Once again, the usefulness of arrays will become more apparent once you have knowledge of for and while loops.

PHP - While Loop

Repetitive tasks are always a burden to us. Deleting spam email, sealing 50 envelopes, and going to work are all examples of tasks that are repeated. The nice thing about programming is that you can avoid such repetitive tasks with a little bit of extra thinking. Most often these repetitive tasks are conquered in the *loop*.

The idea of a loop is to do something over and over again until the task has been completed. Before we show a real example of when you might need one, let's go over the structure of the PHP while loop.

Simple While Loop Example

The function of the while loop is to do a task over and over as long as the specified conditional statement is *true*. This logical check is the same as the one that appears in a PHP if statement to determine if it is *true* or *false*. Here is the basic structure of a PHP while loop:

Pseudo PHP Code:

```
while ( conditional statement is true){
        //do this code;
}
```

This isn't valid PHP code, but it displays how the while loop is structured. Here is the break down of how a while loop functions when your script is executing:

1. The conditional statement is checked. If it is true, then (2) occurs. If it is false, then (4) occurs.
2. The code within the while loop is executed.
3. The process starts again at (1). Effectively "looping" back.
4. If the conditional statement is false, then the code within is not executed and there is no more looping. The code following the while loop is then executed like normal.

A Real While Loop Example

Imagine that you are running an art supply store. You would like to print out the price chart for number of brushes and total cost. You sell brushes at a flat rate, but would like to display how much different quantities would cost. This will save your customers from having to do the mental math themselves.

You know that a while loop would be perfect for this repetitive and boring task. Here is how to go about doing it.

Pseudo PHP Code:

```
$brush_price = 5;
$counter = 10;

echo "<table border=\"1\" align=\"center\">";
echo "<tr><th>Quantity</th>";
echo "<th>Price</th></tr>";
while ( $counter <= 100 ) {
        echo "<tr><td>";
        echo $counter;
        echo "</td><td>";
        echo $brush_price * $counter;
        echo "</td></tr>";
        $counter = $counter + 10;
}
echo "</table>";
```

Display:

Quantity	Price
10	50
20	100
30	150
40	200
50	250
60	300
70	350
80	400
90	450
100	500

Pretty neat, huh? The loop created a new table row and its respective entries for each quantity, until our counter variable grew past the size of 100. When it grew past 100 our conditional statement failed and the loop stopped being used. Let's review what is going on.

1. We first made a $brush_price and $counter variable and set them equal to our desired values.
2. The table was set up with the beginning table tag and the table headers.
3. The while loop *conditional statement* was checked, and $counter (10) was indeed smaller or equal to 100.
4. The code inside the while loop was executed, creating a new table row for the price of 10 brushes.
5. We then added 10 to $counter to bring the value to 20.
6. The loop started over again at step 3, until $counter grew larger than 100.
7. After the loop had completed, we ended the table.

You may have noticed that we placed slashes infront the quotations in the first echo statement. You have to place slashes before quotations if you do not want the quotation to act as the end of the echo statement. This is called escaping a character and it is discussed in our PHP Strings lesson.

With proper use of loops you can complete large tasks with great ease.

PHP - For Loop

The for loop is simply a while loop with a bit more code added to it. The common tasks that are covered by a for loop are:

1. Set a counter variable to some initial value.
2. Check to see if the conditional statement is true.
3. Execute the code within the loop.
4. Increment a counter at the end of each iteration through the loop.

The *for loop* allows you to define these steps in one easy line of code. It may seem to have a strange form, so pay close attention to the syntax used!

For Loop Example

Let us take the example from the while loop lesson and see how it could be done in a for loop. The basic structure of the for loop is as follows:

Pseudo PHP Code:

```
for ( initialize a counter; conditional statement; increment a counter){
        do this code;
}
```

Notice how all the steps of the loop are taken care of in the *for loop* statement. Each step is separated by a semicolon: initiliaze counter, conditional statement, and the counter increment. A semicolon is needed because these are separate expressions. However, notice that a semicolon is not needed after the "increment counter" expression.

Here is the example of the brush prices done with a *for loop* .

PHP Code:

```
$brush_price = 5;

echo "<table border=\"1\" align=\"center\">";
echo "<tr><th>Quantity</th>";
echo "<th>Price</th></tr>";
for ( $counter = 10; $counter <= 100; $counter += 10) {
        echo "<tr><td>";
        echo $counter;
        echo "</td><td>";
        echo $brush_price * $counter;
        echo "</td></tr>";
}
echo "</table>";
```

Display:

Quantity	Price
10	50
20	100
30	150
40	200
50	250
60	300
70	350
80	400
90	450
100	500

It is important to note that both the *for loop* and *while loop* implementation of the price chart table are both OK at getting the job done. However, the for loop is somewhat more compact and would be preferable in this situation. In later lessons we will see where the *while loop* should be used instead of the for loop.

PHP For Each Loop

Imagine that you have an associative array that you want to iterate through. PHP provides an easy way to use every element of an array with the *Foreach* statement.

In plain english this statement will do the following:

- For each item in the specified array execute this code.

While a For Loop and While Loop will continue until some condition fails, the *For Each* loop will continue until it has gone through every item in the array.

PHP For Each: Example

We have an associative array that stores the names of people in our company as the keys with the values being their age. We want to know how old everyone is at work so we use a *Foreach* loop to print out everyone's name and age.

PHP Code:

```
$employeeAges;
$employeeAges["Lisa"] = "28";
$employeeAges["Jack"] = "16";
$employeeAges["Ryan"] = "35";
$employeeAges["Rachel"] = "46";
$employeeAges["Grace"] = "34";

foreach( $employeeAges as $key => $value){
        echo "Name: $key, Age: $value <br />";
}
```

Display:

```
Name: Lisa, Age: 28
Name: Jack, Age: 16
Name: Ryan, Age: 35
Name: Rachel, Age: 46
Name: Grace, Age: 34
```

The syntax of the *foreach* statement is a little strange, so let's talk about it some.

Foreach Syntax: $something as $key => $value

This crazy statement roughly translates into: For each element of the $employeeAges associative array I want to refer to the *key* as $key and the *value* as $value.

The operator "=>" represents the relationship between a *key* and *value*. You can imagine that the key points => to the value. In our example we named the *key* $key and the *value* $value. However, it might be easier to think of it as $name and $age. Below our example does this and notice how the output is identical because we only changed the variable names that refer to the keys and values.

PHP Code:

```
$employeeAges;
$employeeAges["Lisa"] = "28";
$employeeAges["Jack"] = "16";
$employeeAges["Ryan"] = "35";
$employeeAges["Rachel"] = "46";
$employeeAges["Grace"] = "34";

foreach( $employeeAges as $name => $age){
        echo "Name: $name, Age: $age <br />";
}
```

Display:

```
Name: Lisa, Age: 28
Name: Jack, Age: 16
Name: Ryan, Age: 35
Name: Rachel, Age: 46
Name: Grace, Age: 34
```

PHP - Do While Loop

A "do while" loop is a slightly modified version of the while loop. If you recal from one of the previous lessons on While Loops the conditional statement is checked comes back true then the code within the while loop is executed. If the conditional statement is false then the code within the loop is not executed.

On the other hand, a do-while loop *always* executes its block of code at least once. This is because the conditional statement is not checked until *after* the contained code has been executed.

PHP - While Loop and Do While Loop Contrast

A simple example that illustrates the difference between these two loop types is a conditional statement that is always false. First the while loop:

PHP Code:

```
$cookies = 0;
while($cookies > 1){
        echo "Mmmmm...I love cookies! *munch munch munch*";
}
```

Display:

As you can see, this while loop's conditional statement failed (0 is not greater than 1), which means the code within the while loop was not executed. Now, can you guess what will happen with a do-while loop?

PHP Code:

```
$cookies = 0;
do {
        echo "Mmmmm...I love cookies! *munch munch munch*";
} while ($cookies > 1);
```

Display:

```
Mmmmm...I love cookies! *munch munch munch*
```

The code segment "Mmmm...I love cookies!" was executed even though the conditional statement was false. This is because a do-while loop first *do's* and secondly checks the while condition!

Chances are you will not need to use a do while loop in most of your PHP programming, but it is good to know it's there!

PHP - POST & GET

Recall from the PHP Forms Lesson where we used an HTML form and sent it to a PHP web page for processing. In that lesson we opted to use the the *post* method for submitting, but we could have also chosen the *get* method. This lesson will review both transferring methods.

POST - Review

In our PHP Forms Lesson we used the *post* method. This is what the pertinent line of HTML code looked like:

HTML Code Excerpt:

```
<form action="process.php" method="post">
<select name="item">
...
<input name="quantity" type="text" />
```

This HTML code specifies that the form data will be submitted to the "process.php" web page using the POST method. The way that PHP does this is to store all the "posted" values into an *associative array* called "$_POST". Be sure to take notice the names of the form data names, as they represent the *keys* in the "$_POST" associative array.

Now that you know about associative arrays, the PHP code from "process.php" should make a litte more sense.

PHP Code Excerpt:

```
$quantity = $_POST['quantity'];
$item = $_POST['item'];
```

The form names are used as the *keys* in the associative array, so be sure that you never have two input items in your HTML form that have the same name. If you do, then you might see some problems arise.