Tutorial Overview

This tutorial is aimed at the PHP novice and will teach you PHP from the ground up. If you want a drive-through PHP tutorial this probably is not the right tutorial for you.

Remember, you should not try to plow through this tutorial in one sitting. Read a couple lessons, take a break, then do some more after the information has had some time to sink in.

`http://www.tizag.com/phpT/index.php`

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!

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 multiline 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.

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.

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.

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.

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 ($ 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.

PHP - File Write: Overwriting

Now that *testFile.txt* contains some data we can demonstrate what happens when you open an existing file for writing. All the data contained in the file is wiped clean and you start with an empty file. In this example we open our existing file *testFile.txt* and write some new data into it.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'w') or die("can't open file");
$stringData = "Floppy Jalopy\n";
fwrite($fh, $stringData);
$stringData = "Pointy Pinto\n";
fwrite($fh, $stringData);
fclose($fh);
```

If you now open the *testFile.txt* file you will see that Bobby and Tracy have both vanished, as we expected, and only the data we just wrote is present.

Contents of the testFile.txt File:

```
Floppy Jalopy
Pointy Pinto
```

In the next lesson we will show you how to get information out of a file by using PHP's read data functions!

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'r');
$theData = fread($fh, filesize($myFile));
fclose($fh);
echo $theData;
```

Display:

```
Floppy Jalopy Pointy Pinto
```

Note: It is all on one line because our "testFile.txt" file did not have a
 tag to create an HTML line break. Now the entire contents of the *testFile.txt* file is stored in the string variable *$theData*.

PHP - File Read: gets Function

PHP also lets you read a line of data at a time from a file with the *gets* function. This can or cannot be useful to you, the programmer. If you had separated your data with new lines then you could read in one segment of data at a time with the *gets* function.

Lucky for us our "testFile.txt" file is separated by new lines and we can utilize this function.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'r');
$theData = fgets($fh);
fclose($fh);
echo $theData;
```

testFile.txt Contents:

```
Floppy Jalopy
```

The *fgets* function searches for the first occurrence of "\n" the newline character. If you did not write newline characters to your file as we have done in File Write, then this function might not work the way you expect it to.

PHP - File Truncate

As we have mentioned before, when you open a file for writing with the paramater 'w' it completely wipes all data from that file. This action is also referred to as "truncating" a file. Truncate literally means to shorten.

PHP - File Open: Truncate

To erase all the data from our *testFile.txt* file we need to open the file for normal writing. All existing data within *testFile.txt* will be lost.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'w');
fclose($fh);
```

PHP - Truncate: Why Use It?

Truncating is most often used on files that contain data that will only be used for a short time, before needing to be replaced. These type of files are most often referred to as *temporary* files.

For example, you could create an online word processor that automatically saves every thirty seconds. Every time it saves it would take all the data that existed within some HTML form text box and save it to the server. This file, say *tempSave.txt*, would be truncated and overwritten with new, up-to-date data every thirty seconds.

This might not be the most efficient program, but it is a nice usage of truncate.

PHP Code:

```
$numberedString = "12345678901234567890123456789012345678901234567890";

$fivePos = strpos($numberedString, "5");
echo "The position of 5 in our string was $fivePos";
$fivePos2 = strpos($numberedString, "5", $fivePos + 1);
echo "<br />The position of the second 5 was $fivePos2";
```

Display:

```
The position of 5 in our string was 4
The position of the second 5 was 14
```

By taking the first match's position of 4 and adding 1 we then asked *strpos* to begin searching after the last match. The string it was actually sesarching after computing the offset wa: **6789012345...** Letting us find the second 5 in the string.

If we use our knowledge of PHP While Loops we can find every single 5 in our string *numberedString* with just a few lines of code.

PHP Code:

```
$numberedString = "12345678901234567890123456789012345678901234567890";
$offset = 0; // initial offset is 0
$fiveCounter = 0;

while($offset = strpos($numberedString, "5", $offset + 1)){
        $fiveCounter++;
        echo "<br />Five #$fiveCounter is at position - $offset";
}
```

PHP Code:

```
$someWords = "Please don't blow me to pieces.";

$wordChunks = explode(" ", $someWords);
for($i = 0; $i < count($wordChunks); $i++){
        echo "Piece $i = $wordChunks[$i] <br />";
}

$wordChunksLimited = explode(" ", $someWords, 4);
for($i = 0; $i < count($wordChunksLimited); $i++){
        echo "Limited Piece $i = $wordChunksLimited[$i] <br />";
}
```

Display:

```
Piece 0 = Please
Piece 1 = don't
Piece 2 = blow
Piece 3 = me
Piece 4 = to
Piece 5 = pieces.
Limited Piece 0 = Please
Limited Piece 1 = don't
Limited Piece 2 = blow
Limited Piece 3 = me to pieces.
```


The limited explosion has 4 pieces (starting from 0, ending at 3). If you forgot how a for loop works, check out PHP For Loops.

PHP Date - Supplying a Timestamp

As our first example shows, the first argument of the *date* function tells PHP how you would like your date and time displayed. The second argument allows for a timestamp and is optional.

This example uses the *mktime* function to create a timestamp for tomorrow. To go one day in the future we simply add one to the day argument of *mktime*. For your future reference, we have the arguments of *mktime*.

Note: These arguments are all optional. If you do not supply any arguments the current time will be used to create the timestamp.

- mktime(hour, minute, second, month, day, year, daylight savings time)

PHP Code:

```
<?php
$tomorrow = mktime(0, 0, 0, date("m"), date("d")+1, date("y"));
echo "Tomorrow is ".date("m/d/y", $tomorrow);
?>
```

Notice that we used one letter at a time with the function *date* to get the month, day and year. For example the *date("m")* will return the month's number 01-12.

If we were to run our new script just after the 2010 Winter Olympics our display would look like:

Display:

```
Tomorrow is 02/28/10
```


PHP Date - Reference

Now that you know the basics of using PHP's *date* function, you can easily plug in any of the following letters to format your timestamp to meet your needs.

Important Full Date and Time:

- **r**: Displays the full date, time and timezone offset. It is equivalent to manually entering *date("D, d M Y H:i:s O")*

Time:

- **a**: am or pm depending on the time
- **A**: AM or PM depending on the time
- **g**: Hour without leading zeroes. Values are 1 through 12.
- **G**: Hour in 24-hour format without leading zeroes. Values are 0 through 23.
- **h**: Hour with leading zeroes. Values 01 through 12.
- **H**: Hour in 24-hour format with leading zeroes. Values 00 through 23.
- **i**: Minute with leading zeroes. Values 00 through 59.
- **s**: Seconds with leading zeroes. Values 00 through 59.

```
        echo "You've got some stale cookies!";


echo "Your last visit was - ". $visit;
?>
```

This handy script first uses the *isset* function to be sure that our "lastVisit" cookie still exists on the user's PC, if it does, then the user's last visit is displayed. If the user visited our site on February 28, 2008 it might look something like this:

Display:

```
Your last visit was - 11:48 - 02/28/08
```