tizag.com

A PHP Tutorial For Beginners

Written and edited by Tizag.com staff

www.tizag.com

unlock your potential!

Contents

PHP Tutorial - Learn PHP

If you want to learn the basics of PHP, then you've come to the right place. The goal of this tutorial is to teach you the basics of PHP so that you can:

- .:. Customize PHP scripts that you download, so that they better fit your needs.
- .:. Begin to understand the working model of PHP, so you may begin to design your own PHP projects.
- .:. Give you a solid base in PHP, so as to make you more valuable in the eyes of future employers.

PHP stands for PHP Hypertext Preprocessor.

PHP - What is it?

Taken directly from PHP's home, PHP.net, "PHP is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in. The goal of the language is to allow web developers to write dynamically generated pages quickly."

This is generally a good definition of PHP. However, it does contain a lot of terms you may not be used to. Another way to think of PHP is a powerful, behind the scenes scripting language that your visitors won't see!

When someone visits your PHP webpage, your web server processes the PHP code. It then sees which parts it needs to show to visitors(content and pictures) and hides the other stuff(file operations, math calculations, etc.) then translates your PHP into HTML. After the translation into HTML, it sends the webpage to your visitor's web browser.

PHP - What's it do?

It is also helpful to think of PHP in terms of what it can do for you. PHP will allow you to:

- .:. Reduce the time to create large websites.
- .:. Create a customized user experience for visitors based on information that you have gathered from them.
- .:. Open up thousands of possibilities for online tools. Check out PHP - HotScripts for examples of the great things that are possible with PHP.
- .:. Allow creation of shopping carts for e-commerce websites.

What You Should Know

Before starting this tutorial it is important that you have a basic understanding and experience in the following:

- .:. HTML - Know the syntax and especially HTML Forms.
- .:. Basic programming knowledge - This isn't required, but if you have any traditional programming experience it will make learning PHP a great deal easier.

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.

PHP - Necessary Setup

To begin working with PHP you must first have access to either of the following:

.:. A web hosting account that supports the use of PHP web pages and grants you access to MySQL databases. If you do not have a host, but are interested in signing up for one, we recommend that you first read our Web Host Guide to educate yourself about web hosting and avoid getting ripped off.
.:. Have PHP and MySQL installed on your own computer. Read this lesson thorougly for more information on installing PHP.

Although MySQL is not absolutely necessary to use PHP, MySQL and PHP are wonderful complements to one another and some topics covered in this tutorial will require that you have MySQL access.

Installing PHP

For those who are experienced enough to do this yourself, simply head over to PHP.net - Downloads and download the most recent version of PHP.

However, if you are like most of us, you will most likely want to follow a guide to installing PHP onto your computer. These guides are kindly provided by PHP.net based on the operating system that you are using.

.:. PHP - Windows - Windows Installation Guide
.:. PHP - Mac - Mac Installation Guide
.:. PHP - Linux - Linux Installation Guide

Installing MySQL

As we mentioned before, MySQL is not a requirement to use PHP, however they often go hand in hand.

Visit MySQL's MySQL Installation Guidefor help on installing MySQL.

PHP Installation Troubles

If you have any installation troubles you may visit online communities for help on this common problem.

.:. PHP Builder - A web forum for PHP users.

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 (e.g. helloworld.php) 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!

Note: Also, variable names are case-sensitive, so use the exact same capitalization when using a variable. The variables **$a_number** and **$A_number** are different variables in PHP's eyes.

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:

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 backslash. To escape a quote just place a backslash 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 backslash in front of it (\"). The backslash 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 to increment 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

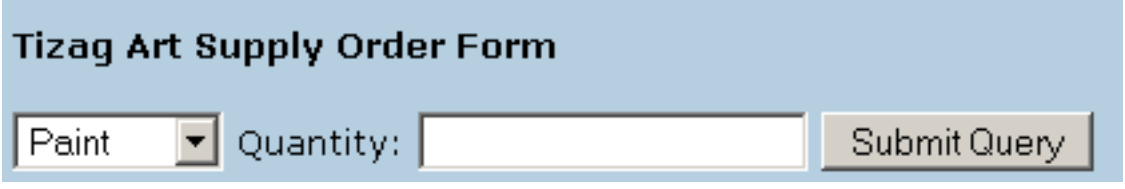
If you need a refresher on how to properly make an HTML form, check out theHTML Form Lesson before continuing on.

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:



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 associative 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>";
```

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>";
```

Quantity	Price
10	50

Quantity	Price
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){
 <indent>echo "Name: $name, Age: $age <br />";</indent>
}
```

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.

PHP - GET

As we mentioned before, the alternative to the *post* method is *get*. If we were to change our HTML form to the *get* method, it would look like this:

HTML Code Excerpt:

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

The *get* method is different in that it passes the variables along to the "process.php" web page by

appending them onto the end of the URL. The URL, after clicking submit, would have this added on to the end of it:

"?item=##&quantity=##"

The question mark "?" tells the browser that the following items are variables. Now that we changed the method of sending information on "order.html", we must change the "process.php" code to use the "$_GET" associative array.

PHP Code Excerpt:

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

After changing the array name the script will function properly. Using the *get* method displays the variable information to your visitor, so be sure you are not sending password information or other sensitive items with the *get* method. You would not want your visitors seeing something they are not supposed to!

Security Precautions

Whenever you are taking user input and using you need to be sure that the input is safe. If you are going to insert the data into a MySQL database, then you should be sure you have thought about preventing MySQL Injection. If you are going to make a user's input available to the public, then you should think about PHP htmlentities.

PHP - Magic Quotes

Prior to PHP 6 there was a feature called *magic quotes* that was created to help protect newbie programmers from writing bad form processing code. *Magic quotes* would automatically escape risky form data that might be used for SQL Injection with a backslash \. The characters escaped by PHP include: quote ', double quote ", backslash \ and NULL characters.

However, this newbie protection proved to cause more problems than it solved and is not in PHP 6. If your PHP version is any version before 6 then you should use this lesson to learn more about how magic quotes can affect you.

Magic Quotes - Are They Enabled?

First things first, you need to check to see if you have magic quotes enabled on you server. The *get_magic_quotes_gpc* function will return a 0 (off) or a 1 (on). These boolean values will fit nicely into an *if statement* where 1 is true and 0 is false.

PHP Code:

```
if(get_magic_quotes_gpc())
 echo "Magic quotes are enabled";
else
 echo "Magic quotes are disabled";
```

Display:

```
Magic quotes are enabled
```

If you received the message "Magic quotes are enabled" then you should definitely continue reading this lesson, if not feel free to learn about it in case you are developing for servers that might have quotes on or off.

Magic Quotes in Action

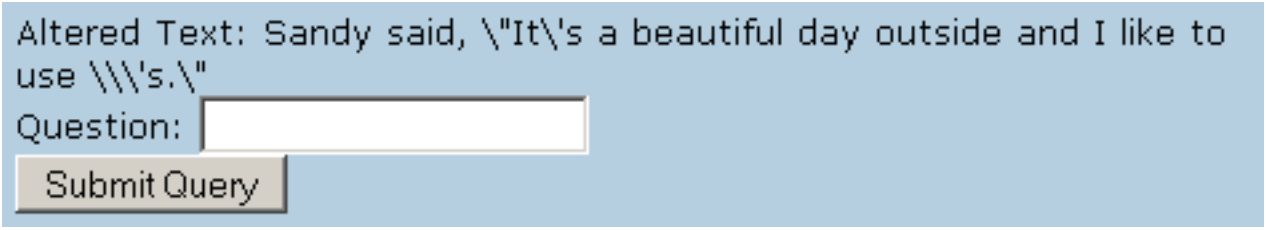
Now lets make a simple form processor to show how machines with magic quotes enabled will escape those potentially risky characters. This form submits to itself, so you only need to make one file, "magic-quotes.php" to test it out.

magic-quotes.php Code:

```
<?php
echo "Altered Text: ".$_POST['question'];
?>

<form method='post'>
Question: <input type='text' name='question'/><br />
<input type='submit'>

</form>
```

This simple form will display to you what magic quotes is doing. If you were to enter and submit the string: Sandy said, "It's a beautiful day outside and I like to use \'s." You would receive the following output.

Display:



Magic quotes did a number on that string, didn't it? Notice that there is a backslash before all of those risky characters we talked about earlier. After magic quotes:

- **.:.** A backslash \ becomes \\
- **.:.** A quote ' becomes \'
- **.:.** A double-quote " becomes \"

Now say that you wanted to remove the escaping that magic quotes puts in, you have two options: disable magic quotes or strip the backslashes magic quotes adds.

Removing Backslashes - stripslashes()

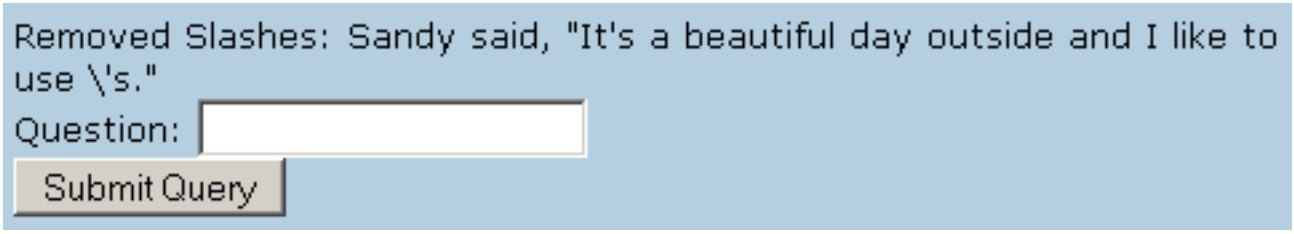
Before you use PHP's backslash removal function *stripslashes* it's smart to add some magic quote checking like our "Are They Enabled?" section above. This way you won't accidentally be removing slashes that are legitimate in the future if your PHP's magic quotes setting changes in the future.

magic-quotes.php Code:

```
<?php
echo "Removed Slashes: ";
// Remove those slashes
if(get_magic_quotes_gpc())
 echo stripslashes($_POST['question']);
else
 echo $_POST['question'];

?>

<form method='post'>
Question: <input type='text' name='question'/><br />
<input type='submit'>

</form>
```

Our new output for our string containing risky characters would now be:

Display:



PHP htmlentities Function

Whenever you allow your users to submit text to your website, you need to be careful that you don't leave any security holes open for malicious users to exploit. If you are ever going to allow user submitted text to be visible by the public you should consider using the *htmlentities* function to prevent them from running html code and scripts that may be harmful to your visitors.

PHP - Converting HTML into Entities

The *htmlentities* function takes a string and returns the same string with HTML converted into HTML entities. For example, the string "<script>" would be converted to "<script>".

By converting the < and > into entities, it prevents the browser from using it as an HTML element and it prevents the code from running if you were to display some user's input on your website.

This may seem a little complicated, but if you think of the way a browser works, in separate stages, it becomes a little easier. Let's look at the way the function *htmlentities* changes the data at three different levels: in PHP, in raw HTML and in the web browser. The sample string is a bad script that will redirect visitors to the malicious user's own website.

PHP Code:

```
// An imaginary article submission from a bad user
//  it will redirect anyone to example.com if the code is run in a browser
$userInput = "I am going to hax0r your site, hahaha!
 <script type='text/javascript'>
 window.location = 'http://www.example.com/'
 </script>'";

//Lets make it safer before we use it
$userInputEntities = htmlentities($userInput);

//Now we can display it
echo $userInputEntities;
```

The HTML output of the above script would be as follows:

Safe Raw HTML Code:

```
I am going to hax0r your site, hahaha!
 &lt;script type='text/javascript'&gt;
 window.location = 'http://www.example.com/'
 &lt;/script&gt;'
```

If we had not used *htmlentities* to convert any HTML code into safe entities, this is what the raw HTML code would be and it would have redirect a visitor to example.com.

Dangerous Raw HTML Code:

```
I am going to hax0r your site, hahaha!
 <script type='text/javascript'>
 window.location = 'http://www.example.com/'
 </script>'
```

Those two HTML code examples are what you would see if you were to view source on the web page. However, if you were just viewing the output normally in your browser you would see the following.

Safe Display:

```
I am going to hax0r your site, hahaha!
 <script type='text/javascript'>
 window.location = 'http://www.example.com/'
 </script>'
```

Dangerous Display:

```
You'd see whatever spammer site that the malicious user had sent you to.
Probably
some herbal supplement site or weight loss pills would be displayed.
```

When Would You Use htmlentities?

Anytime you allow users to submit content to your website, that other visitors can see, you should consider removing the ability to let them use HTML. Although this will remove a lot of cool things that your users can do, like making heavily customized content, it will prevent your site from a lot of common attacks. With some custom coding you can just remove specific tags from running, but that is beyond the scope of this lesson.

Just remember, that when allowing users to submit content to your site you are also giving them access to your website. Be sure you take the proper precautions.

PHP - Files

Manipulating files is a basic necessity for serious programmers and PHP gives you a great deal of tools for creating, uploading, and editing files.

This section of the PHP tutorial is completely dedicated to how PHP can interact with files. After completing this section you should have a solid understanding of all types of file manipulation in PHP!

PHP - Files: Be Careful

When you are manipulating files you must be very careful because you can do a lot of damage if you do something wrong. Common errors include editing the wrong file, filling a hard-drive with garbage data, and accidentally deleting a file's contents.

It is our hope that you will be able to avoid these and other slipups after reading this tutorial. However, we know that there are so many places where code can take a wrong turn, so we urge you to take extra care when dealing with files in PHP.

PHP - Files: Overview

The presentation of the file lessons will begin with how to create, open, and close a file. After establishing those basics, we will then cover other important file tasks, such as: read, write, append, truncate, and uploading files with PHP.

PHP - File Create

Before you can do anything with a file it has to exist! In this lesson you will learn how to create a file using PHP.

PHP - Creating Confusion

In PHP, a file is created using a command that is also used to open files. It may seem a little confusing, but we'll try to clarify this conundrum.

In PHP the *fopen* function is used to open files. However, it can also **create** a file if it does not **find** the file specified in the function call. So if you use *fopen* on a file that does not exist, it will create it, given that you open the file for writing or appending (more on this later).

PHP - How to Create a File

The *fopen* function needs two important pieces of information to operate correctly. First, we must supply it with the name of the file that we want it to open. Secondly, we must tell the function what we plan on doing with that file (i.e. read from the file, write information, etc).

Since we want to create a file, we must supply a file name and tell PHP that we want to write to the file. Note: We have to tell PHP we are writing to the file, otherwise it will not create a new file.

PHP Code:

```
$ourFileName = "testFile.txt";
$ourFileHandle = fopen($ourFileName, 'w') or die("can't open file");
fclose($ourFileHandle);
```

The file "testFile.txt" should be created in the same directory where this PHP code resides. PHP will see that "testFile.txt" does not exist and will create it after running this code. There's a lot of information in those three lines of code, let's make sure you understand it.

1. **$ourFileName = "testFile.txt";**
   Here we create the name of our file, "testFile.txt" and store it into a PHP String variable *$ourFileName*.

4. **$ourFileHandle = fopen($ourFileName, 'w')** or die("can't open file");
   This bit of code actually has two parts. First we use the function *fopen* and give it two arguments: our file name and we inform PHP that we want to write by passing the character "w".

   Second, the *fopen* function returns what is called a *file handle*, which will allow us to manipulate the file. We save the file handle into the *$ourFileHandle* variable. We will talk more about file handles later on.

9. **fclose($ourFileHandle);**
   We close the file that was opened. *fclose* takes the file handle that is to be closed. We will talk more about this more in the file closing lesson.

PHP - Permissions

If you are trying to get this program to run and you are having errors, you might want to check that you have granted your PHP file access to write information to the hard drive. Setting permissions is most often done with the use of an FTP program to execute a command called *CHMOD*. Use CHMOD to allow the PHP file to write to disk, thus allowing it to create a file.

In the near future Tizag.com will have a more in-depth tutorial on how to use CHMOD to set file permissions.

PHP - File Open

In the previous lesson we used the function *fopen* to create a new file. In this lesson we will be going into the details of this important function and see what it has to offer.

PHP - Different Ways to Open a File

For many different technical reasons, PHP requires you to specify your intentions when you open a file. Below are the three basic ways to open a file and the corresponding character that PHP uses.

.:. Read: 'r'

Open a file for read only use. The file pointer begins at the front of the file.

.:. Write: 'w'

Open a file for write only use. In addition, the data in the file is erased and you will begin writing data at the **beginning** of the file. This is also called truncating a file, which we will talk about more in a later lesson. The file pointer begins at the start of the file.

.:. Append: 'a'

Open a file for write only use. However, the data in the file is preserved and you begin will writing data at the **end** of the file. The file pointer begins at the end of the file.

A *file pointer* is PHP's way of remembering its location in a file. When you open a file for reading, the file pointer begins at the start of the file. This makes sense because you will usually be reading data from the front of the file.

However, when you open a file for appending, the file pointer is at the end of the file, as you most likely will be appending data at the end of the file. When you use reading or writing functions they begin at the location specified by the file pointer.

PHP - Explanation of Different Types of fopen

These three basic ways to open a file have distinct purposes. If you want to get information out of a file, like search an e-book for the occurrences of "cheese", then you would open the file for read only.

If you wanted to write a new file, or overwrite an existing file, then you would want to open the file with the "w" option. This would wipe clean all existing data within the file.

If you wanted to add the latest order to your "orders.txt" file, then you would want to open it to append the data on to the end. This would be the "a" option.

PHP - File Open: Advanced

There are additional ways to open a file. Above we stated the standard ways to open a file. However, you can open a file in such a way that reading and writing is allowable! This combination is done by placing a plus sign "+" after the file mode character.

.:. Read/Write: 'r+'

Opens a file so that it can be read from and written to. The file pointer is at the beginning of the file.

.:. Write/Read: 'w+'

This is exactly the same as r+, **except** that it deletes all information in the file when the file is opened.

.:. Append: 'a+'

This is exactly the same as r+, **except** that the file pointer is at the end of the file.

PHP - File Open: Cookie Cutter

Below is the correct form for opening a file with PHP. Replace the (X) with one of the options above (i.e. r, w, a, etc).

Pseudo PHP Code:

```
$ourFileName = "testFile.txt";
$fh = fopen($ourFileName, 'X') or die("Can't open file");
fclose($fh);
```

PHP - File Open: Summary

You can open a file in many different ways. You can delete everything and begin writing on a clean slate, you can add to existing data, and you can simply read information from a file. In later lessons we will go into greater detail on how each of these different ways to open a file is used in the real world and give some helpful examples.

PHP - File Close

The next logical step after you have opened a file and finished your business with it is to close that file down. You don't want an open file running around on your server taking up resources and causing mischief!

PHP - File Close Description

In PHP it is not system critical to close all your files after using them because the server will close all files after the PHP code finishes execution. **However** the programmer is still free to make mistakes (i.e. editing a file that you accidentally forgot to close). You should close all files after you have finished with them because it's a good programming practice and because we told you to!

PHP - File Close Function

In a previous tutorial, we had a call to the function *fclose* to close down a file after we were done with it. Here we will repeat that example and discuss the importance of closing a file.

PHP Code:

```
$ourFileName = "testFile.txt";
$ourFileHandle = fopen($ourFileName, 'w') or die("can't open file");
fclose($ourFileHandle);
```

The function *fclose* requires the file handle that we want to close down. In our example we set our variable "$fileHandle" equal to the file handle returned by the *fopen* function.

After a file has been closed down with *fclose* it is impossible to read, write or append to that file unless it is once more opened up with the *fopen* function.

PHP - File Write

Now that you know how to open and close a file, lets get on to the most useful part of file manipulation, writing! There is really only one main function that is used to write and it's logically called *fwrite*.

PHP - File Open: Write

Before we can write information to our test file we have to use the function *fopen* to open the file for writing.

PHP Code:

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

PHP - File Write: fwrite Function

We can use php to write to a text file. The *fwrite* function allows data to be written to any type of file. Fwrite's first parameter is the file handle and its second parameter is the string of data that is to be written. Just give the function those two bits of information and you're good to go!

Below we are writing a couple of names into our test file *testFile.txt* and separating them with a carriaged return.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'w') or die("can't open file");
$stringData = "Bobby Bopper\n";
fwrite($fh, $stringData);
$stringData = "Tracy Tanner\n";
fwrite($fh, $stringData);
fclose($fh);
```

The *$fh* variable contains the file handle for *testFile.txt*. The file handle knows the current file pointer, which for writing, starts out at the beginning of the file.

We wrote to the file *testFile.txt* twice. Each time we wrote to the file we sent the string *$stringData* that first contained *Bobby Bopper* and second contained *Tracy Tanner*. After we finished writing we closed the file using the *fclose* function.

If you were to open the *testFile.txt* file in NOTEPAD it would look like this:

Contents of the testFile.txt File:

```
Bobby Bopper
```

```
Tracy Tanner
```

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 - File Read

My apologies for taking so long to actually get to the point where you get information from files. In this lesson we will teach you how to read data from a file using various PHP functions.

PHP - File Open: Read

Before we can read information from a file we have to use the function *fopen* to open the file for reading. Here's the code to read-open the file we created in the PHP File Write lessons.

PHP Code:

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

The file we created in the last lesson was named "testFile.txt". Your PHP script that you are writing should reside in the same directory as "text.txt". Here are the contents of our file from File Write.

testFile.txt Contents:

```
Floppy Jalopy
Pointy Pinto
```

Now that the file is open, with read permissions enabled, we can get started!

PHP - File Read: fread Function

The *fread* function is the staple for getting data out of a file. The function requires a file handle, which we have, and an integer to tell the function how much data, in bytes, it is supposed to read.

One character is equal to one byte. If you wanted to read the first five characters then you would use five as the integer.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'r');
$theData = fread($fh, 5);
fclose($fh);
echo $theData;
```

Display:

```
Flopp
```

The first five characters from the *testFile.txt* file are now stored inside *$theData*. You could echo this string, *$theData*, or write it to another file.

If you wanted to read all the data from the file, then you need to get the size of the file. The *filesize* function returns the length of a file, in bytes, which is just what we need! The *filesize* function requires the name of the file that is to be sized up.

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 Delete

You know how to create a file. You know how to open a file in an assortment of different ways. You even know how to read and write data from a file!

Now it's time to learn how to **destroy** (delete) files. In PHP you delete files by calling the *unlink* function.

PHP - File Unlink

When you view the contents of a directory you can see all the files that exist in that directory because the operating system or application that you are using displays a list of filenames. You can think of these filenames as links that join the files to the directory you are currently viewing.

If you unlink a file, you are effectively causing the system to forget about it or delete it!

Before you can delete (unlink) a file, you must first be sure that it is not open in your program. Use the *fclose* function to close down an open file.

PHP - Unlink Function

Remember from the PHP File Create lesson that we created a file named *testFile.txt*.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'w') or die("can't open file");
fclose($fh);
```

Now to delete *testFile.txt* we simply run a PHP script that is located in the same directory. Unlink just needs to know the name of the file to start working its destructive magic.

PHP Code:

```
$myFile = "testFile.txt";
unlink($myFile);
```

The *testFile.txt* should now be removed.

PHP - Unlink: Safety First!

With great power comes a slough of potential things you can mess up! When you are performing the unlink function be sure that you are deleting the right file!

PHP - File Append

So far we have learned how to open, close, read, and write to a file. However, the ways in which we have written to a file so far have caused the data that was stored in the file to be deleted. If you want to *append* to a file, that is, add on to the existing data, then you need to open the file in append mode.

PHP - File Open: Append

If we want to add on to a file we need to open it up in append mode. The code below does just that.

PHP Code:

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

If we were to write to the file it would begin writing data at the end of the file.

PHP - File Write: Appending Data

Using the *testFile.txt* file we created in the File Write lesson , we are going to append on some more data.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'a') or die("can't open file");
$stringData = "New Stuff 1\n";
fwrite($fh, $stringData);
$stringData = "New Stuff 2\n";
fwrite($fh, $stringData);
fclose($fh);
```

You should noticed that the way we write data to the file is exactly the same as in the Write lesson. The only thing that is different is that the file pointer is placed at the end of the file in append mode, so all data is added to the end of the file.

The contents of the file *testFile.txt* would now look like this:

Contents of the testFile.txt File:

```
Floppy Jalopy
Pointy Pinto
New Stuff 1
New Stuff 2
```

PHP - Append: Why Use It?

The above example may not seem very useful, but appending data onto a file is actually used everyday. Almost all web servers have a *log* of some sort. These various logs keep track of all kinds of information, such as: errors, visitors, and even files that are installed on the machine.

A log is basically used to document events that occur over a period of time, rather than all at once. Logs: a perfect use for append!

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 - File Upload

A very useful aspect of PHP is its ability to manage file uploads to your server. Allowing users to upload a file to your server opens a whole can of worms, so please be careful when enabling file uploads.

PHP - File Upload: HTML Form

Before you can use PHP to manage your uploads, you must first build an HTML form that lets users select a file to upload. See our HTML Form lesson for a more in-depth look at forms.

HTML Code:

```
<form enctype="multipart/form-data" action="uploader.php" method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Choose a file to upload: <input name="uploadedfile" type="file" /><br
/>
<input type="submit" value="Upload File" />
</form>
```

Here is a brief description of the important parts of the above code:

- **.:. enctype="multipart/form-data"** - Necessary for our to-be-created PHP file to function properly.
- **.:. action="uploader.php"** - The name of our PHP page that will be created, shortly.
- **.:. method="POST"** - Informs the browser that we want to send information to the server using POST.
- **.:. input type="hidden" name="MA...** - Sets the maximum allowable file size, in bytes, that can be uploaded. This safety mechanism is easily bypassed and we will show a solid backup solution in PHP. We have set the max file size to 100KB in this example.
- **.:. input name="uploadedfile"** - *uploadedfile* is how we will access the file in our PHP script.

Save that form code into a file and call it *upload.html*. If you view it in a browser it should look like this:

Display:

After the user clicks submit, the data will be posted to the server and the user will be redirected to *uploader.php*. This PHP file is going to process the form data and do all the work.

PHP - File Upload: What's the PHP Going to Do?

Now that we have the right HTML form we can begin to code the PHP script that is going to handle our uploads. Typically, the PHP file should make a key decision with **all** uploads: keep the file or throw it away. A file might be thrown away from many reasons, including:

- **.:.** The file is too large and you do not want to have it on your server.
- **.:.** You wanted the person to upload a picture and they uploaded something else, like an executable file

(.exe).
.:. There were problems uploading the file and so you can't keep it.

This example is very simple and omits the code that would add such functionality.

PHP - File Upload: uploader.php

When the *uploader.php* file is executed, the uploaded file exists in a temporary storage area on the server. If the file is not moved to a different location it will be **destroyed**! To save our precious file we are going to need to make use of the *$_FILES* associative array.

The *$_FILES* array is where PHP stores all the information about files. There are two elements of this array that we will need to understand for this example.

.:. uploadedfile - *uploadedfile* is the reference we assigned in our HTML form. We will need this to tell the $_FILES array which file we want to play around with.
.:. $_FILES['uploadedfile']['name'] - *name* contains the original path of the user uploaded file.
.:. $_FILES['uploadedfile']['tmp_name'] - *tmp_name* contains the path to the temporary file that resides on the server. The file should exist on the server in a temporary directory with a temporary name.

Now we can finally start to write a basic PHP upload manager script! Here is how we would get the temporary file name, choose a permanent name, and choose a place to store the file.

PHP Code:

```
// Where the file is going to be placed
$target_path = "uploads/";

/* Add the original filename to our target path.
Result is "uploads/filename.extension" */
$target_path = $target_path . basename( $_FILES['uploadedfile']['name']);
$_FILES['uploadedfile']['tmp_name'];
```

NOTE: You will need to create a new directory in the directory where *uploader.php* resides, called "uploads", as we are going to be saving files there.

We now have all we need to successfully save our file to the server. *$target_path* contains the path where we want to save our file to.

PHP - File Upload: move_uploaded_file Function

Now all we have to do is call the *move_uploaded_file* function and let PHP do its magic. The *move_uploaded_file* function needs to know 1) The path of the temporary file (check!) 2) The path where it is to be moved to (check!).

PHP Code:

```
$target_path = "uploads/";

$target_path = $target_path . basename( $_FILES['uploadedfile']['name']);

if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target_path)) {
    echo "The file ".  basename( $_FILES['uploadedfile']['name']).
    " has been uploaded";
} else{
    echo "There was an error uploading the file, please try again!";
}
```

If the upload is successful, then you will see the text "The file *filename* has been uploaded". This is because *$move_uploaded_file* returns *true* if the file was moved, and *false* if it had a problem.

If there was a problem then the error message "There was an error uploading the file, please try again!" would be displayed.

PHP - File Upload: Safe Practices!

Note: This script is for education purposes only. We do not recommend placing this on a web page viewable to the public.

These few lines of code we have given you will allow anyone to upload data to your server. Because of this, we recommend that you do not have such a simple file uploader available to the general public. Otherwise, you might find that your server is filled with junk or that your server's security has been compromised.

We hope you enjoyed learning about how to work with uploading files with PHP. In the near future we will be adding an advanced lesson that will include more security and additional features!

PHP - String Position - strpos

Being able to manipulate strings is a valuable skill, especially in PHP. You'll most likely come across a programming problem that requires you to find some data in a string. The beginning of a lot of your string manipulation expertise will begin with the *strpos* function, which allows you to find data in your string.

Searching a String with strpos

The way *strpos* works is it takes some string you want to search in as its first argument and another string, which is what you are actually searching for, as the second argument. If the function can find a search match, then it will return the position of the **first** match. However, if it can't find a match it will return *false*.

To make this function crystal clear, lets search a numbered, in-order string, for the number five.

PHP Code:

```
$numberedString = "1234567890"; // 10 numbers from 1 to 0

$fivePos = strpos($numberedString, "5");
echo "The position of 5 in our string was $fivePos";
```

Display:

```
The position of 5 in our string was 4
```

Notice that the position is 4, which may seem confusing at first, until you realize that PHP starts counting from 0.

- The number 1 - Position 0 - No match
- The number 2 - Position 1 - No match
- The number 3 - Position 2 - No match
- The number 4 - Position 3 - No match
- The number 5 - Position 4 - **Match**

Although we only searched for a single character, you can use this function to search for a string with any number of characters. Also, it is important to note that this function will return the position of the *start* of the first match. So if we had searched the same string for "567890" we would again find a match and position 4 because that is where the match starts.

Finding All Occurrences in a String with Offset

One of the limitations of *strpos* is that it only returns the position of the very first match. If there are 5,000 other matches in the string you would be none the wiser, unless you take action!

There is a third (optional) argument to *strpos* that will let you specify where to begin your search of

the string. If you were to store the position of the last match and use that + 1 as an offset, you would skip over the first match and be find the next one.

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";
}
```

Display:

```
Five #1 is at position - 4
Five #2 is at position - 14
Five #3 is at position - 24
Five #4 is at position - 34
```

That conditional statement in our while loop may look a little intimidating, but not if you break it down.

.:. $offset = strpos($numberedString, "5", $offset + 1) - This is our conditional statement for our PHP While Loop. If this ever is *false* the while loop will stop running. This conditional statement always runs before each pass through the while loop.

.:. strpos($numberedString, "5", $offset + 1) - This is the same code we used in a previous example. We are going to search our string *numberedString* for the number 5 and use the last match's value (stored in $offset) + 1 to skip over the last match. The first $offset we use has a value of 0, so that we start at the beginning of the string.
.:. $offset = strpos(... We are going to store the location returned by *strpos* into $offset so that we can skip this match the next time the while loop runs through the code. If *strpos* ever fails to find a match then this will be set to *false* making our while loop stop executing.

PHP str_replace Function

Another key tool to have in your programming toolbox is the ability to quickly replace parts of a PHP string with new values. The *str_replace* function is similar to a word processor's "Replace All" command that lets you specify a word and what to replace it with, then replaces every occurrence of that word in the document.

str_replace Parameters

str_replace has three parameters that are required for the function to work properly. *str_replace(search, replace, originalString)*.

1. **search -** This is what you want to search your string for. This can be a string or an array.
2. **replace** - All matches for *search* will be replaced with this value. This can be a string or an array.
3. **originalString** - This is what search and replace will be operating on. The *str_replace* function will return a modified version of originalString when it completes.

str_replace Simple Example

Imagine we are working at a school district and need to create a webpage for the students' parents. The webpage has an introduction string that we need to customize depending on if the student is male or female. With *str_replace* this is mighty easy.

PHP Code:

```
//string that needs to be customized
$rawstring = "Welcome Birmingham parents. Your replaceme is a pleasure to
have!";

//male string
$malestr = str_replace("replaceme", "son", $rawstring);

//female string
$femalestr = str_replace("replaceme", "daughter", $rawstring);

echo "Son: ". $malestr . "<br />";
echo "Daughter: ". $femalestr;
```

Display:

```
Son: Welcome Birmingham parents. Your son is a pleasure to have!
Daughter: Welcome Birmingham parents. Your daughter is a pleasure to have!
```

With these two gender customized strings created we could then provide a more engaging experience for the student's parents when they logged into the school website with their kid's credentials.

str_replace Arrays: Multiple Replaces in One

In the last example we only needed to replace one word *replaceme* in our string, but what if we wanted to replace many words? We could just use the function multiple times to get the job done, **or** we could create an array of *placeholders* and a second array of *replace values* to get it all done in one function call.

The key thing to understand with this technique is that you are creating two arrays that will be used to swap values. The first item in *placeholders* will be replaced by the first item in the *replace values*, the second item of *placeholders* replaced with the second in *replace values* and so on and so forth.

Let's extend our simple example to be a complete form letter addressed to a student's parents.

PHP Code:

```
//string that needs to be customized
$rawstring = "Welcome Birmingham parent! <br />
 Your offspring is a pleasure to have!
 We believe pronoun is learning a lot.<br />
 The faculty simple adores pronoun2 and you can often hear
 them say \"Attah sex!\"<br />";

//placeholders array
$placeholders = array('offspring', 'pronoun', 'pronoun2', 'sex');
//male replace values array
$malevals = array('son', 'he', 'him', 'boy');
//female replace values array
$femalevals = array('daughter', 'she', 'her', 'girl');

//male string
$malestr = str_replace($placeholders, $malevals, $rawstring);

//female string
$femalestr = str_replace($placeholders, $femalevals, $rawstring);

echo "Son: ". $malestr . "<br />";
echo "Daughter: ". $femalestr;
```

Display:

```
Son: Welcome Birmingham parent!
Your son is a pleasure to have! We believe he is learning a lot.
The faculty simple adores he2 and you can often hear them say "Attah boy!"

Daughter: Welcome Birmingham parent!
Your daughter is a pleasure to have! We believe she is learning a lot.
The faculty simple adores she2 and you can often hear them say "Attah girl!"
```

Notice: there is a bug in this code. The placeholder *pronoun2* did not get replaced in the way we intended (our strings have he2 and she2 instead of him and her). This is because all instances of *pronoun* were replaced first and the pronoun in pronoun2 was replaced at this time with he or she, making he2 or she2. When it was *pronoun2*'s turn to be replaced, there were no matches to be found, so our string has no him or her.

To fix this bug you could simply make sure that pronoun2 comes first in the *placeholders* array and by updating the values of the male and female replace values to reflect this.

PHP Code:

```
// ...snip
//placeholders array
$placeholders = array('offspring', 'pronoun2', 'pronoun',  'sex');
//male replace values array
$malevals = array('son', 'him', 'he',  'boy');
//female replace values array
$femalevals = array('daughter', 'her', 'she',  'girl');
//snip...
```

Display:

```
Son: Welcome Birmingham parent!
Your son is a pleasure to have! We believe he is learning a lot.
The faculty simple adores him and you can often hear them say "Attah boy!"

Daughter: Welcome Birmingham parent!
Your daughter is a pleasure to have! We believe she is learning a lot.
The faculty simple adores her and you can often hear them say "Attah girl!"
```

PHP substr_replace Function

The function *substr_replace* introduces some additional functionality to compliment *str_replace*. *substr_replace* is a more mathematically based replace function, which relies on starting points and lengths to replace parts of strings, as opposed to searching and replacing.

substr_replace's Four Parameters

There are three required parameters for the *substr_replace* function (*original string*, *replacement string*, *starting point*) and one that's optional (*length*).

1. **original string** - This is your original string that will be operated on.
2. **replacement string** - This string will be used to replace everything in the string from the *starting point* to the ending point (specified by *length*).
3. **starting point** - This is the place in the *original string* that will be used to mark the replacement's beginning. A negative value specifies the number of characters from the end of the string.
4. optional **length** - How many characters from the original string will be replaced. If no length is specified then the end of the string is used. If a value of 0 is used then no characters will be replaced and an *insert* is performed. A negative value specifies the number of characters from the end of the string.

substr_replace On Your Mark

This example of *substr_replace* shows what happens when you omit the *length* parameter at various *starting points*.

PHP Code:

```
//string that needs to be customized
$original = "ABC123 Hello Mr. Cow! DEF321";

//starting point 5
$sp5 = substr_replace($original, "Five", 5);
//starting point 12
$sp12 = substr_replace($original, "Twelve", 12);
//starting point 0
$sp0 = substr_replace($original, "Zero", 0);
//starting point -1
$spneg1 = substr_replace($original, "Negative 1", -1);

//Echo each string
echo "Original String: $original <br />";
echo "Starting Point 5: $sp5 <br />";
echo "Starting Point 12: $sp12 <br />";
echo "Starting Point 0: $sp0 <br />";
echo "Starting Point -1: $spneg1 ";
```

Display:

```
Original String: ABC123 Hello Mr. Cow! DEF321
Starting Point 5: ABC12Five
Starting Point 12: ABC123 HelloTwelve
Starting Point 0: Zero
Starting Point -1: ABC123 Hello Mr. Cow! DEF32Negative 1
```

As you can see, when you don't specify the fourth parameter, *length*, everything after the starting point is replaced by the second parameter *replacement string*.

Note: The first replacement occurred at position 5, which in *$original* was the character 3. This 3 and everything onward was replaced with the *replacement string*. Remember that you start counting character to begin from zero. The $original string could be labeled as so:

- Letter A - Position 0
- Letter B - Position 1
- Letter C - Position 2
- Letter 1 - Position 3
- Letter 2 - Position 4
- Letter 3 - Position 5

substr_replace Specifying a Length

If you want to get any sort of precision out of this function you're going to have to get into the nitty gritty of specifying the exact *length* of characters you want replaced in your *original string*.

Imagine that you want to get rid of those ugly pseudo references (ABC123, DEF321) at the beginning and end of the string. Since both of those strings are a length of 6 and we know one is at the very beginning of the string and the other is at the very end of the string we should probably use a starting point of 0 for ABC123 and a value of -6 for DEF321. By having a *replacement string* of nothing "" we can do something similar to select and delete that we often do in a word processor.

PHP Code:

```
//string that needs to be customized
$original = "ABC123 Hello Mr. Cow! DEF321";

//remove ABC123 and store in $cleanedstr
$cleanedstr = substr_replace($original, "", 0, 6);
//remove DEF321 from $cleanedstr
$cleanedstr2 = substr_replace($cleanedstr, "", -6, 6);

//Echo each string
echo "Original String: $original <br />";
echo "Clean #1: $cleanedstr <br />";
echo "Clean #2: $cleanedstr2";
```

Display:

```
Original String: ABC123 Hello Mr. Cow! DEF321
Clean #1: Hello Mr. Cow! DEF321
Clean #2: Hello Mr. Cow!
```

Make sure that you play around with this function some on your own so you can get a feel for how the *starting point* and *length* parameters effect this function.

substr_replace Perform an Insert

By setting the *length* parameter to zero you can stop *substr_replace* from removing anything from the original string and just add to it. If we wanted to add a second and third person to our *$original* string we would want to do this insert operation. **Note**: instead of counting the characters we've used a couple other PHP functions to figure out the *starting positions* for us.

PHP Code:

```
//string that needs to be customized
$original = "Hello Mr. Cow!";

// Get the position of Mr. Cow
$cowpos = strpos($original, "Mr. Cow");

// Find where Mr. Cow ends by adding the length of Mr. Cow
$cowpos_end = $cowpos + strlen("Mr. Cow");

// Insert Mrs. Bear after Mr. Cow
$mrsbear = substr_replace($original, " and Mrs. Bear", $cowpos_end, 0);

// Insert Sensei Shark before Mr. Cow
$senseishark = substr_replace($mrsbear, "Sensei Shark, ", $cowpos, 0);


//Echo each string
echo "Original String: $original <br />";
echo "After Mrs. Bear: $mrsbear <br />";
echo "After Sensei Shark: $senseishark";
```

Display:

```
Original String: Hello Mr. Cow!
After Mrs. Bear: Hello Mr. Cow and Mrs. Bear!
After Sensei Shark: Hello Sensei Shark, Mr. Cow and Mrs. Bear!
```

We snuck a new function **strlen** into that example, but it isn't that complicated of a function, as it stands for "string length."

.:. $cowpos_end = $cowpos + strlen("Mr. Cow");

The **strlen** function takes a string and then counts up how many characters are in it then returns

that number. So by calculating the length of "Mr. Cow" and adding that to the position, we find out where the end point is!

PHP - String Capitalization Functions

If you've ever wanted to manipulate the capitalization of your PHP strings, then this lesson will be quite helpful to you. PHP has three primary capitalization related functions: strtoupper, strtolower and ucwords. The function names are pretty self-explanatory, but why they are useful in programming might be new to you.

Converting a String to Upper Case - strtoupper

The *strtoupper* function takes one argument, the string you want converted to upper case and returns the converted string. Only letters of the alphabet are changed, numbers will remain the same.

PHP Code:

```
$originalString = "String Capitalization 1234";

$upperCase = strtoupper($originalString);
echo "Old string - $originalString <br />";
echo "New String - $upperCase";
```

Display:

```
Old string - String Capitalization 1234
New String - STRING CAPITALIZATION 1234
```

One might use this function to increase emphasis of a important point or in a title. Another time it might be used with a font that looks very nice with all caps to fit the style of the web page design.

A more technical reason would be to convert two strings you are comparing to see if they are equal. By converting them to the same capitalization you remove the possibility that they won't match simply because of different capitalizations.

Converting a String to Lower Case - strtolower

The *strtolower* function also has one argument: the string that will be converted to lower case.

PHP Code:

```
$originalString = "String Capitalization 1234";

$lowerCase = strtolower($originalString);
echo "Old string - $originalString <br />";
echo "New String - $lowerCase";
```

Display:

```
Old string - String Capitalization 1234
New String - string capitalization 1234
```

Capitalizing the First Letter - ucwords

Titles of various media types often capitalize the first letter of each word and PHP has a time-saving function that will do just this.

PHP Code:

```
$titleString = "a title that could use some hELP";

$ucTitleString = ucwords($titleString);
echo "Old title - $titleString <br />";
echo "New title - $ucTitleString";
```

Display:

```
Old title - a title that could use some hELP
New title - A Title That Could Use Some HELP
```

Notice that the last word "hELP" did not have the capitalization changed on the letters that weren't first, they remained capitalized. If you want to ensure that **only** the first letter is capitalized in each word of your title, first use the *strtolower* function and then the *ucwords* function.

PHP Code:

```
$titleString = "a title that could use some hELP";

$lowercaseTitle = strtolower($titleString);
$ucTitleString = ucwords($lowercaseTitle);
echo "Old title - $titleString <br />";
echo "New title - $ucTitleString";
```

Display:

```
Old title - a title that could use some hELP
New title - A Title That Could Use Some Help
```

PHP - String Explode

The PHP function *explode* lets you take a string and blow it up into smaller pieces. For example, if you had a sentence you could ask *explode* to use the sentence's spaces " " as dynamite and it would blow up the sentence into separate words, which would be stored in an array. The sentence **"Hello, I would like to lose weight."** would look like this after explode got done with it:

1. Hello,
2. I
3. would
4. like
5. to
6. lose
7. weight.

The dynamite (the space character) disappears, but the other stuff remains, but in pieces. With that abstract picture of the *explode* function in mind, lets take a look at how it really works.

The explode Function

The first argument that *explode* takes is the delimiter (our dynamite) which is used to blow up the second argument, the original string. *explode* returns an array of string pieces from the original and they are numbered in order, starting from 0. Lets take a phone number in the form ###-###-#### and use a hyphen "-" as our dynamite to split the string into three separate chunks.

PHP Code:

```
$rawPhoneNumber = "800-555-5555";

$phoneChunks = explode("-", $rawPhoneNumber);
echo "Raw Phone Number = $rawPhoneNumber <br />";
echo "First chunk = $phoneChunks[0]<br />";
echo "Second chunk = $phoneChunks[1]<br />";
echo "Third Chunk chunk = $phoneChunks[2]";
```

Display:

```
Raw Phone Number = 800-555-5555
First chunk = 800
Second chunk = 555
Third Chunk chunk = 5555
```

explode Function - Setting a Limit

If you want to control the amount of destruction that explode can wreak on your original string, consider using the third (optional) argument which allows you to set the number of pieces explode can return. This means it will stop exploding once the number of pieces equals the set limit. Below we've blown up a sentence with no limit and then with a limit of 4.

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 - Array implode

The PHP function *implode* operates on an array and is known as the "undo" function of *explode*. If you have used *explode* to break up a string into chunks or just have an array of stuff you can use *implode* to put them all into one string.

PHP implode - Repairing the Damage

The first argument of implode is the string of characters you want to use to join the array pieces together. The second argument is the array (pieces).

PHP Code:

```
$pieces = array("Hello", "World,", "I", "am", "Here!");

$gluedTogetherSpaces = implode(" ", $pieces);
$gluedTogetherDashes = implode("-", $pieces);
for($i = 0; $i < count($pieces); $i++){
 echo "Piece #$i = $pieces[$i] <br />";
}
echo "Glued with Spaces = $gluedTogetherSpaces <br />";
echo "Glued with Dashes = $gluedTogetherDashes";
```

Display:

```
Piece #0 = Hello
Piece #1 = World,
Piece #2 = I
Piece #3 = am
Piece #4 = Here!
Glued with Spaces = Hello World, I am Here!
Glued with Dashes = Hello-World,-I-am-Here!
```

The *implode* function will convert the entire array into a string and there is no optional argument to limit this as there was in the *explode* function.

PHP Date - Robust Dates and Times

While PHP's *date()* function may seem to have an overwhelming amount of options available, isn't it always better to have more choices than not enough? With PHP's date function you format timestamps, so they are more human readable.

This lesson will teach you how to display the current time, formating PHP's timestamp, and show you all the various date arguments for reference purposes.

PHP Date - The Timestamp

The date function always formats a timestamp, whether you supply one or not. What's a timestamp? Good question!

.:. Timestamp: A timestamp is the number of seconds from January 1, 1970 at 00:00. Otherwise known as the Unix Timestamp, this measurement is a widely used standard that PHP has chosen to utilize.

PHP Date - What Time Is It?

The date function uses letters of the alphabet to represent various parts of a typical date and time format. The letters we will be using in our first example are:

.:. d: The day of the month. The type of output you can expect is 01 through 31.
.:. m: The current month, as a number. You can expect 01 through 12.
.:. y: The current year in two digits ##. You can expect 00 through 99

We'll tell you the rest of the options later, but for now let's use those above letters to format a simple date! The letters that PHP uses to represent parts of date and time will automatically be converted by PHP.

However, other characters like a slash "/" can be inserted between the letters to add additional formatting. We have opted to use the slash in our example.

PHP Code:

```
<?php
echo date("m/d/y");
?>
```

If the 2010 Winter Olympics were just finishing up, you would see something like:

Display:

```
02/27/10
```

Be sure to test this out on your own PHP enabled server, it's really great to see the instant results

available with PHP date!

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.

Day:

.:. **d**: Day of the month with leading zeroes. Values are 01 through 31.

- **j**: Day of the month without leading zeroes. Values 1 through 31
- **D**: Day of the week abbreviations. Sun through Sat
- **l**: Day of the week. Values Sunday through Saturday
- **w**: Day of the week without leading zeroes. Values 0 through 6.
- **z**: Day of the year without leading zeroes. Values 0 through 365.

Month:

- **m**: Month number with leading zeroes. Values 01 through 12
- **n**: Month number without leading zeroes. Values 1 through 12
- **M**: Abbreviation for the month. Values Jan through Dec
- **F**: Normal month representation. Values January through December.
- **t**: The number of days in the month. Values 28 through 31.

Year:

- **L**: 1 if it's a leap year and 0 if it isn't.
- **Y**: A four digit year format
- **y**: A two digit year format. Values 00 through 99.

Other Formatting:

- **U**: The number of seconds since the Unix Epoch (January 1, 1970)
- **O**: This represents the Timezone offset, which is the difference from Greenwich Meridian Time (GMT). 100 = 1 hour, -600 = -6 hours

We suggest that you take a few minutes to create several timestamps using PHP's *mktime* function and just try out all these different letters to get your feet wet with PHP's *date* function.

PHP Sessions - Why Use Them?

As a website becomes more sophisticated, so must the code that backs it. When you get to a stage where your website need to pass along user data from one page to another, it might be time to start thinking about using PHP sessions.

A normal HTML website will not pass data from one page to another. In other words, all information is forgotten when a new page is loaded. This makes it quite a problem for tasks like a shopping cart, which requires data(the user's selected product) to be remembered from one page to the next.

PHP Sessions - Overview

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping cart items, etc). However, this session information is temporary and is usually deleted very quickly after the user has **left** the website that uses sessions.

It is important to ponder if the sessions' temporary storage is applicable to your website. If you require a more permanent storage you will need to find another solution, like a MySQL database.

Sessions work by creating a unique identification(UID) number for each visitor and storing variables based on this ID. This helps to prevent two users' data from getting confused with one another when visiting the same webpage.

Note:If you are not experienced with session programming it is not recommended that you use sessions on a website that requires high-security, as there are security holes that take some advanced techniques to plug.

Starting a PHP Session

Before you can begin storing user information in your PHP session, you must first start the session. When you start a session, it must be at the very beginning of your code, before any HTML or text is sent.

Below is a simple script that you should place at the beginning of your PHP code to start up a PHP session.

PHP Code:

```
<?php
session_start(); // start up your PHP session!
?>
```

This tiny piece of code will register the user's session with the server, allow you to start saving user information and assign a UID (unique identification number) for that user's session.

Storing a Session Variable

When you want to store user data in a session use the $_SESSION associative array. This is where

you both store and retrieve session data. In previous versions of PHP there were other ways to perform this store operation, but it has been updated and this is the correct way to do it.

PHP Code:

```
<?php
session_start();
$_SESSION['views'] = 1; // store session data
echo "Pageviews = ". $_SESSION['views']; //retrieve data
?>
```

Display:

```
Pageviews = 1
```

In this example we learned how to store a variable to the session associative array $_SESSION and also how to retrieve data from that same array.

PHP Sessions: Using PHP's *isset* Function

Now that you know can easily store and retrieve data from the $_SESSION array, we can now explore some of the real functionality of sessions. When you create a variable and store it in a session, you probably want to use it in the future. However, before you use a session variable it is necessary that you check to see if it exists already!

This is where PHP's *isset* function comes in handy. *isset* is a function that takes any variable you want to use and checks to see if it has been **set**. That is, it has already been assigned a value.

With our previous example, we can create a very simple pageview counter by using *isset* to check if the pageview variable has already been created. If it has we can increment our counter. If it doesn't exist we can create a pageview counter and set it to one. Here is the code to get this job done:

PHP Code:

```
<?php
session_start();
if(isset($_SESSION['views']))
    $_SESSION['views'] = $_SESSION['views']+ 1;
else
    $_SESSION['views'] = 1;

echo "views = ". $_SESSION['views'];
?>
```

The first time you run this script on a **freshly opened browser** the *if statement* will fail because no session variable *views* would have been stored yet. However, if you were to refresh the page the *if statement* would be true and the counter would increment by one. Each time you reran this script you would see an increase in *view* by one.

Cleaning and Destroying your Session

Although a session's data is temporary and does not require that you explicitly clean after yourself, you may wish to delete some data for your various tasks.

Imagine that you were running an online business and a user used your website to buy your goods. The user has just completed a transaction on your website and you now want to remove everything from their shopping cart.

PHP Code:

```
<?php
session_start();
if(isset($_SESSION['cart']))
    unset($_SESSION['cart']);
?>
```

You can also completely destroy the session entirely by calling the *session_destroy* function.

PHP Code:

```
<?php
session_start();
session_destroy();
?>
```

Destroy will reset your session, so don't call that function unless you are entirely comfortable losing all your stored session data!

PHP Cookies - Background

Cookies have been around for quite some time on the internet. They were invented to allow webmaster's to store information about the user and their visit on the user's computer.

At first they were feared by the general public because it was believed they were a serious privacy risk. Nowadays nearly everyone has cookies enabled on their browser, partly because there are worse things to worry about and partly because all of the "trustworthy" websites now use cookies.

This lesson will teach you the basics of storing a cookie and retrieving a cookie, as well as explaining the various options you can set with your cookie.

Creating Your First PHP Cookie

When you create a cookie, using the function *setcookie*, you must specify three arguments. These arguments are ***setcookie****(name, value, expiration)*:

1. **name**: The name of your cookie. You will use this name to later retrieve your cookie, so don't forget it!
2. **value**: The value that is stored in your cookie. Common values are username(string) and last visit(date).
3. **expiration**: The date when the cookie will expire and be deleted. If you do not set this expiration date, then it will be treated as a session cookie and be removed when the browser is restarted.

In this example we will be creating a cookie that stores the user's last visit to measure how often people return to visit our webpage. We want to ignore people that take longer than two months to return to the site, so we will set the cookie's expiration date to two months in the future!

PHP Code:

```
<?php
//Calculate 60 days in the future
//seconds * minutes * hours * days + current time
$inTwoMonths = 60 * 60 * 24 * 60 + time();
setcookie('lastVisit', date("G:i - m/d/y"), $inTwoMonths);
?>
```

Don't worry if you can't follow the somewhat involved date calculations in this example. The important part is that you know how to set a cookie, by specifying the three important arguments: name, value and expiration date.

Retrieving Your Fresh Cookie

If your cookie hasn't expired yet, let's retrieve it from the user's PC using the aptly named *$_COOKIE* associative array. The name of your stored cookie is the key and will let you retrieve your stored cookie value!

PHP Code:

```
<?php
if(isset($_COOKIE['lastVisit']))
 $visit = $_COOKIE['lastVisit'];
else
 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
```