# PHP Tutorials

**What is PHP, and why do I need it?**

PHP is probably the most popular scripting language on the web. It is used to enhance web pages. With PHP, you can do things like create username and password login pages, check details from a form, create forums, picture galleries, surveys, and a whole lot more. If you've come across a web page that ends in PHP, then the author has written some programming code to liven up the plain, old HTML.

PHP is known as a server-sided language. That's because the PHP doesn't get executed on your computer, but on the computer you requested the page from. The results are then handed over to you, and displayed in your browser. Other scripting languages you may have heard of are ASP, Python and Perl. (You don't need to know any of these to make a start on PHP. In fact, these tutorials assume that you have no programming experience at all.)

The most popular explanation of just what PHP stands for is "Hypertext Pre-processor". But that would make it HPP, surely? An alternative explanation is that the initials come from the earliest version of the program, which was called Personal Home Page Tools. At least you get the letters "PHP" in the right order!

But PHP is so popular that if you're looking for a career in the web design/web scripting industry then you just have to know it! In these tutorials, we'll get you up and running. And, hopefully, it will be a lot easier than you think.

### Course Files

To follow along with these tutorials, you'll need our Home and Learn Course files. These can be downloaded from the following address:

[Download the PHP Course Files](http://www.homeandlearn.co.uk/downloads.html)

You'll also need to have a server, to test your scripts. Don't worry, though - we've found an easy way to get a server up and running on your own PC. Move on to the next part to learn more.

# PHP Tutorials

### What you need to get started with PHP

Before you can write and test your PHP scripts, there's one thing you'll need - a server! Fortunately, you don't need to go out and buy one. In fact, you won't be spending any extra money. That's why PHP is so popular! But because PHP is a server-sided scripting language, you either have to get some web space with a hosting company that supports PHP, or make your computer pretend that it has a server installed. This is because PHP is not run on your PC - it's executed on the server. The results are then sent back to the client PC (your computer).

Don't worry if this all sounds a little daunting - we've come across an easier way to get you up and running. We're going to be using some software called Wampserver. This allows you to test your PHP scripts on your own computer. It installs everything you need, if you have a Windows PC. We'll explain how to get it installed in a moment, and where to get it from. But just a word for non-windows users.

### Apple Users

If you have OS X, then try these sites to get up and running with PHP:

http://www.onlamp.com/pub/a/mac/2001/12/07/apache.html

http://www.entropy.ch/software/macosx/php/

What you're doing here is getting the apache server up and running, so that you can run PHP scripts offline. Pay particular attention to where files are stored, and to the "localhost" address.

### Linux Users

There are quite a few sites out there to help Linux users get up and running with the Apache server and PHP. Here are three sites that are worth checking out:

http://en.wikipedia.org/wiki/LAMP\_(software\_bundle)

http://www.php-mysql-tutorial.com/wikis/php-tutorial/installing-php-and-mysql.aspx

http://www.phpfreaks.com/tutorials/12/0.php

If you know any better ones, we'd be interested in hearing from you!

### Windows Users

OK, back to Wampserver and Windows. First, you need to download the software. You can get it from here (this site is nothing to do with ours, by the way):

[Download Wampserver](http://www.wampserver.com/en/download.php)

Be sure to click the link for Presentation, as well as the link for Downloads. The Presentation page shows you how to install the file.

# PHP Tutorials

### Installing and Testing Wampserver

Hopefully, you have now downloaded and installed Wampserver. This will give you a server on your own PC (Windows users), somewhere you can test your scripts. If you haven't yet dowloaded the Wampserver software, you can download it here:

[Download Wampserver](http://www.wampserver.com/en/download.php)

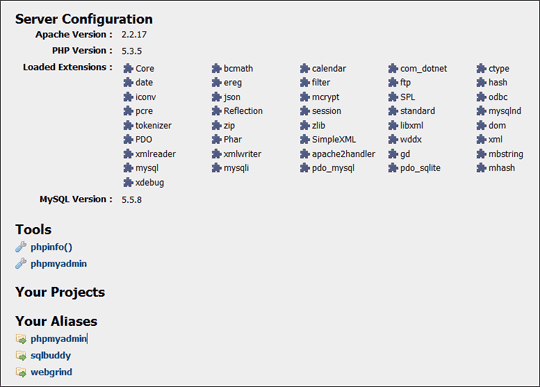
If the installation went well, you should have an new icon in the bottom right, where the clock is:



Click the icon to see the menu above.

From here, you can stop the server, exit it, view help files, and see the configuration pages.

Click on **localhost**, though, and you'll see this page appear: (Localhost just refers to the server running on your own computer. Another way to refer to your server is by using the IP address 127.0.0.1.)



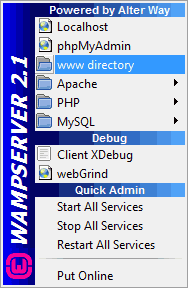
Click the link under **Tools** that says **phpinfo**(). If all went well, you should be looking at the following page (The one below is a different php version, but don't worry about this - as long as you see something):

[The info.php page](http://www.homeandlearn.co.uk/php/php1p3.html) (click to open in a new window 66K)

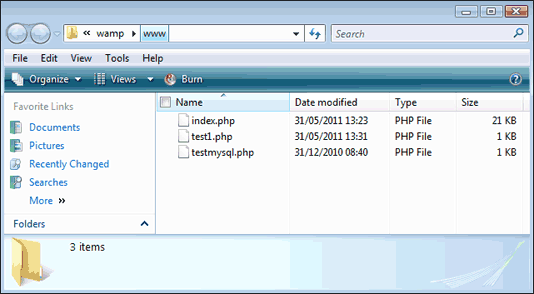
If you saw the above page, then congratulations! Your PHP server is up and running, and you can make a start scripting PHP pages.

### Saving your PHP files

Whenever you create a new PHP page, you need to save it in your WWW directory. You can see where this is by clicking its item on the menu:



When you click on www directory, you should see an explorer window appear. This one is from Windows Vista: (You'll probably have only two files, index and testmysql.)



This www folder for Wampserver is usally at this location on your hard drive:

**c:/wamp/www/**

Bear this in mind when you click **File > Save As** to save your PHP scripts.

### Launching your PHP scripts

Suppose you have created a php script called **test1.php**. To launch this script, you need to add the script name after localhost in your browser. So instead of this:

**http://localhost/index.php**

You would type this:

**http://localhost/test1.php**

You don't type the name of the wamp folder, however. This would be wrong, for example:

**c:/wamp/www/test1.php**

As too would this:

**http://localhost/www/test1.php**

Your server knows where the www folder is, so you don't have to type it out: just add the script name to **localhost**. Likewise, if you create a folder under www then you'd just type this:

**http://localhost/folder\_name/script\_name.php**

OK, we'll assume that everything is now up and running. If it's not, click "Move on to the Next Part" below, for some troubleshooting. If it is, click "Back to the PHP Contents Page".

# PHP Tutorials

### Troubleshooting

If you don't see the info.php page, then you will need to refer to the wampserver forums. The page you need is here:

[Wampserver Forums](http://www.wampserver.com/phorum/list.php?2)

Hopefully, your question will already have been asked an answered. (Unfortunately, we can't answer questions about Wampserver, as it's not our software.)

# PHP Tutorials - What is a Variable?

A variable is just a storage area. You put things into your storage areas (variables) so that you can use and manipulate them in your programmes. Things you'll want to store are numbers and text.

If you're ok with the idea of variables, then you can move on. If not, think of them like this. Suppose you want to catalogue your clothing collection. You enlist two people to help you, a man and a woman. These two people are going to be your storage areas. They are going to hold things for you, while you tally up what you own. The man and the woman, then, are variables.

You count how many coats you have, and then give these to the man. You count how many shoes you have, and give these to the woman. Unfortunately, you have a bad memory. The question is, which one of your people (variables) holds the coats and which one holds the shoes? To help you remember, you can give your people names! You could call them something like this:

**mr\_coats**  
**mrs\_shoes**

But it's entirely up to you what names you give your people (variables). If you like, they could be called this:

**man\_coats  
woman\_shoes**

Or

**HimCoats  
HerShoes**

But because your memory is bad, it's best to give them names that help you remember what it is they are holding for you. (There are some things your people balk at being called. You can't begin their names with an underscore (\_), or a number. But most other characters are fine.)

OK, so your people (variables) now have name. But it's no good just giving them a name. They are going to be doing some work for you, so you need to tell them what they will be doing. The man is going to be holding the coats. But we can specify how many coats he will be holding. If you have ten coats to give him, then you do the "telling" like this:

**mr\_coats = 10**

So, the variable name comes first, then an equals sign. After the equals sign, you tell your variable what it will be doing. Holding the number 10, in our case. (The equals sign, by the way, is not really an equals sign. It's called an assignment operator. But don't worry about it, at this stage. Just remember that you need the equals sign to store things in your variables.)

However, you're learning PHP, so there's something missing. Two things, actually. First, your people (variables) need a dollar sign at the beginning (people are like that). So it would be this:

**$mr\_coats = 10**

If you miss the dollar sign out, then your people will refuse to work! But the other thing missing is something really picky and fussy - a semi-colon. Lines of code in PHP need a semi-colon at the end:

**$mr\_coats = 10;**

If you get any parse errors when you try to run your code, the first thing to check is if you've missed the semi-colon off the end. It's very easy to do, and can be frustrating. The next thing to check is if you've missed out a dollar sign. But back to our people (variables).

So the man is holding ten coats. We can do the same thing with the other person (variable):

**$mrs\_shoes = 25;**

So, $mrs\_shoes is holding a value of 25. If we then wanted to add up how many items of clothes we have so far, we could set up a new variable (Note the dollar sign at the begining of the new variable):

**$total\_clothes**

We can then add up the coats and the shoes. You add up in PHP like this:

**$total\_clothes = $mr\_coats + $mrs\_shoes;**

Remember, $mr\_coats is holding a value of 10, and $mrs\_shoes is holding a value of 25. If you use a plus sign, PHP thinks you want to add up. So it will work out the total for you. The answer will then get stored in our new variable, the one we've called $total\_clothes. You can also add up like this:

**$total\_clothes = 10 + 35;**

Again, PHP will see the plus sign and add the two together for you. Of course, you can add up more than two items:

**$total\_clothes = 10 + 35 + 7 + 38 + 1250;**

But the idea is the same - PHP will see plus signs and then add things up. The answer is then stored in your variable name, the one to the left of the equals sign.

In the next part, we'll take a look at how to put text into variables.

# PHP Tutorials - Putting Text into Variables

In the [previous section](http://www.homeandlearn.co.uk/php/php2p1.html), you saw how to put numbers into variables. But you can also put text into your variables. Suppose you want to know something about the coats you own. Are they Winter coats? Jackets? Summer coats? You decide to catalogue this, as well. You can put direct text into your variables. You do it in a similar way to storing numbers:

**$coats1 = "Winter Coats";**

Again, our variable name starts with a dollar sign ($). We've then given it the name coats1. The equals sign follows the variable name. After the equals sign, however, we have direct text - Winter Coats. But notice the double quotation marks around our text. If you don't surround your direct text with quotation marks, then you'll get errors. You can, however, use single quotes instead of double quotes. So you can do this:

**$coats1 = 'Winter Coats';**

But you can't do this:

**$coats1 = 'Winter Coats";**

In the above line, we've started with a single quote and ended with a double quote. This will get you an error.

We can store other text in the same way:

**$coats2 = "Jackets";  
$coats3 = "Summer Coats";**

The direct text will then get stored in the variable to the left of the equals sign.

So, to recap, variables are storage areas. You use these storage areas to manipulate things like text and numbers. You'll be using variables a lot, and on the next few pages you'll see how they work in practice.

# PHP Tutorials: Variables - Some Practice

In the previous section, you saw what variables are: storage areas to hold things like numbers and text. You tell PHP to remember these values because you want to do something with them. In this section, you'll get some practice using variables. Off we go.

### Testing variables with PHP

First, we'll take a look at how to display what's in your variables. We're going to be viewing our results on a web page. So see if you can get this script working first, because it's the one we'll be building on. Using a text editor like Notepad, or your PHP software, type the following. (You can copy and paste it, if you prefer. But you learn more by typing it out yourself - it doesn't really sink in unless you're making mistakes!)

**<html>  
<head>  
<title>Variables - Some Practice</title>  
</head>  
<body>**

**<?php print("It Worked!"); ?>**

**</body>  
</html>**

When you've finished typing it all, save the page as **variables.php**. Then Run the script. Remember: when you're saving your work, save it to the WWW folder, as explained [here](http://www.homeandlearn.co.uk/php/php1p3.html). To run the page, start your browser up and type this in the address bar:

**http://localhost/variables.php**

If you've created a folder inside the www folder, then the address to type in your browser would be something like:

**http://localhost/FolderName/variables.php**

If you were successful, you should have seen the text "It worked!" displayed in your browser. If so, Congratulations! You have a working server up and running! (If you're using Wampserver, you should see an icon in the bottom right of your screen. Click the icon and select **Start All Services** from the menu.)

The PHP script is only one line long:

**<?php print("It Worked!"); ?>**

The rest of the script is just plain HTML code. Let's examine the PHP in more detail.

We've put the PHP in the BODY section of an HTML page. Scripts can also, and often do, go between the HEAD section of an HTML page. You can also write your script without any HTML. But before a browser can recognise your script, it needs some help. You have to tell it what kind of script it is. Browsers recognise PHP by looking for this punctuation (called syntax):

**<?php ?>**

So you need a left angle bracket ( < ) then a question mark ( ? ). After the question mark, type PHP (in upper or lowercase). After your script has finished, type another question mark. Finally, you need a right angle bracket ( > ). You can put as much space as you like between the opening and closing syntax.

To display things on the page, we've used **print( )**. What you want the browser to print goes between the round brackets. If you're printing direct text, then you need the quotation marks (single or double quotes). To print what's inside of a variable, just type the variable name (including the dollar). Finally, the line of code ends as normal - with a semi-colon (;). Another way to display things on the page is to use an alternative to print() – **echo( )**.

Now let's adapt the basic page so that we can set up some variables. We'll try some text first. Keep the HTML as it is, but change your PHP from this:

**<?php print("It Worked!"); ?>**

To this:

**<?php**

**print("It Worked!");**

**?>**

OK, it's not much of a change! But spreading your code out over more than one line makes it easier to see what you're doing. Now, it's clear that there's only one line of code - Print. So add this second line to your code (the one in red):

**<?php**

**$test\_String = "It Worked!";**

**print("It Worked!");**

**?>**

We've set up a variable called **$test\_String**. After the equals sign, the text "It Worked!" has been added. The line is then ended with a semi-colon. Don't run your script yet. Change the Print line to this:

**print($test\_String);**

Then add some comments ...

**<?php  
//--------------TESTING VARIABLES------------**

**$test\_String = "It Worked!";  
print($test\_String);  
?>**

Comments in PHP are for your benefit. They help you remember what the code is supposed to do. A comment can be added by typing two slashes. This tells PHP to ignore the rest of the line. After the two slashes, you can type anything you like. Another way to add a comment, is like this:

**<?php**

**/\* --------------TESTING VARIABLES------------  
Use this type of comment if you want to spill over to more than one line.  
Notice how the comment begin and end.  
\*/**

**$test\_String = "It Worked!";**

**print($test\_String);**

**?>**

Whichever method you choose, make sure you add comment to your code: they really do help. Especially if you have to send your code to someone else!

But you can now run the script above, and test it out.

How did you get on? You should have seen that exactly the same text got printed to the page. And you might be thinking - what's the big deal? Well, what you just did was to pass some text to a variable, and then have PHP print the contents of the variable. It's a big step: your coding career has now begun!

**Exercise**  
Change the text "It Worked!" to anything you like. Then run the script again. Try typing some numbers in between your double quotes, instead of text.

**Exercise**  
Change the double quotes to single quotes. Did it have any effect? Put a single quote at the beginning of your text, and a double quote at the end. What happens when you run the code?

**Exercise**  
Delete the dollar sign from the variable name. Then run your code. What error did you get? Put the dollar sign back, but now delete the semi-colon. Run your code again? What error did you get, this time? It's well worth remembering these errors - you'll see them a lot when you're starting out! If you see them in future, you'll be better able to correct your errors.

Now that you're up and running, we'll do some more variable work in the next section.

# PHP Tutorials - More Variable Practice

In [the previous section](http://www.homeandlearn.co.uk/php/php2p3.html), you started to work with variables. You outputted text to a page. In the next few sections, you'll do some more work with variables, and learn how to do your sums with PHP.

But now that you can print text to a page, let's try some numbers. Start with the basic PHP page again, and save your work as **variables2.php**:

**<html>  
<head>  
<title>More on Variables</title>  
</head>  
<body>**

**<?php**

**print ("Basic Page");**

**?>**

**</body>  
</html>**

We'll now set up a variable and print it to the page. So change your code to this:

**<?php**

**$first\_number = 10;  
print ($first\_number);**

**?>**

All the code does is to print the contents of the variable that we've called **$first\_number**. Remember: if you're printing direct text then you need quotation marks; if you're printing a variable name then you leave the quotes out. To see why, run the first script above. Then change the print line to this:

**print ("$first\_number");**

In other words, add double quotation marks around your variable name. Did it make a difference? What did you expect would print out? Now change the double quotes to single quotes. Run your script again. With double quotes, the number 10 still prints; with single quotes, you get the variable name!

**TIP:** We recommend you use single quotes for your direct text, and NOT double quotes - there's fewer hassles if you do!

# PHP Tutorials - Concatenate

You can join together direct text, and whatever is in your variable. The full stop (period or dot, to some) is used for this. Suppose you want to print out the following "My variable contains the value of 10". In PHP, you can do it like this:

**<?php**

**$first\_number = 10;  
$direct\_text = 'My variable contains the value of ';**

**print ($direct\_text . $first\_number);**

**?>**

So now we have two variables. The new variable holds our direct text. When we're printing the contents of both variables, a full stop is used to separate the two. Try out the above script, and see what happens. Now delete the dot and then try the code again. Any errors?

You can also do this sort of thing:

**<?php**

**$first\_number = 10;**

**print ('My variable contains the value of ' . $first\_number);**

**?>**

This time, the direct text is not inside a variable, but just included in the Print statement. Again a full stop is used to separate the direct text from the variable name. What you've just done is called concatenation. Try the new script and see what happens.

# PHP Tutorials - Adding up in PHP

OK, let's do some adding up. To add up in PHP, the plus symbol (**+**) is used. (If you still have the code open from the previous page, try changing the full stop to a plus symbol. Run the code, and see what happens.)

To add up the contents of variables, you just separate each variable name with a plus symbol. Try this new script:

**<?php**

**$first\_number = 10;  
$second\_number = 20;  
$sum\_total = $first\_number + $second\_number;**

**$direct\_text = 'The two variables added together = ';**

**print ($direct\_text . $sum\_total);**

**?>**

In the above script, we've added a second number, and assigned a value to it:

**$second\_number = 20;**

A third variable is then declared, which we've called **$sum\_tota**l. To the right of the equals sign, we've added up the contents of the first variable and the contents of the second variable:

**$sum\_total = $first\_number + $second\_number;**

PHP knows what is inside of the variables called **$first\_number** and **$second\_number**, because we've just told it in the two line above! It sees the plus symbol, then adds the two values together. It puts the answer to the addition in the variable to the left of the equals sign (=), the one we've called **$sum\_total**.

To print out the answer, we've used concatenation:

**print ($direct\_text . $sum\_total);**

This script is a little more complicated than the ones you've been doing. If you're a bit puzzled, just remember what it is we're doing: adding the contents of one variable to the contents of another. The important line is this one:

**$sum\_total = $first\_number + $second\_number;**

The addition to the right of the equals sign gets calculated first (**$first\_number + $second\_number**). The total of the addition is then stored in the variable to the left of the equals sign (**$sum\_total =**).

You can, of course, add up more than two numbers. Try this exercise.

**Exercise**  
Add a third variable to your code. Assign a value of 30 to your new variable. Put the sum total of all three variables into the variable called **$sum\_total**. Use concatenation to display the results. (In other words, add up 10, 20, and 30!)

You don't have to use variable names to add up. You can do this:

**print (10 + 20 + 30);**

Or even this:

**$number = 10;**

**print ($number + 30);**

But the point is the same - use the plus symbol (+) to add up.

In the few next parts, you'll learn how to Subtract, Divide and Multiply.

# PHP Tutorials - How to Subtract in PHP

We're not going to weigh things down by subjecting you to torrents of heavy Math! But you do need to know how to use the basic operators. First up is subtracting.

To add up using PHP variables, you did this:

**<?php**

**$first\_number = 10;  
$second\_number = 20;  
$sum\_total = $first\_number + $second\_number;**

**print ($sum\_total);**

**?>**

Subtraction is more or less the same. Instead of the plus sign (+), simply use the minus sign (-). Change your **$sum\_total** line to this, and run your code:

**$sum\_total = $second\_number - $first\_number;**

The **s$sum\_total** line is more or less the same as the first one. Except we're now using the minus sign instead (and reversing the two variables). When you run the script you should, of course, get the answer 10. Again, PHP knows what is inside of the variables called **$second\_number** and **$first\_number**. It knows this because you assigned values to these variables in the first two lines. When PHP comes across the minus sign, it does the subtraction for you, and puts the answer into the variable on the left of the equals sign. We then use a print statement to display what is inside of the variable.

Just like addition, you can subtract more than one number at a time. Try this:

**<?php**

**$first\_number = 10;  
$second\_number = 20;  
$third\_number = 100;**

**$sum\_total = $third\_number - $second\_number - $first\_number;**

**print ($sum\_total);**

**?>**

The answer you should get is 70. You can also mix addition with subtraction. Here's an example:

**<?php**

**$first\_number = 10;  
$second\_number = 20;  
$third\_number = 100;**

**$sum\_total = $third\_number - $second\_number + $first\_number;**

**print ($sum\_total);**

**?>**

Run the code above. What answer did you get? Was it the answer you were expecting? Why do you think it printed the number it did? If you thought it might have printed a different answer to the one you got, the reason might be the way we set out the sum. Did we mean 100 - 20, and then add the 10? Or did we mean add up 10 and 20, then take it away from 100? The first sum would get 90, but the second sum would get 70.

To clarify what you mean, you can use parentheses in your sums. Here's the two different versions of the sum. Try them both in your code. But note where the parentheses are:

**Version one**  
$sum\_total = **(**$third\_number - $second\_number**)** + $first\_number;

**Version two**  
$sum\_total = $third\_number - **(**$second\_number + $first\_number**)**;

It's always a good idea to use parentheses in your sums, just to clarify what you want PHP to calculate. That way, you won't get a peculiar answer!

Another reason to use parentheses is because of something called **operator precedence**. In PHP, some operators (Math symbols) are calculated before others. This means that you'll get answers that are entirely unexpected! As we'll find out right now in the next part - Multiplication.

# PHP Tutorials - Multiplication

To multiply in PHP (and just about every other programming language), the **\*** symbol is used. If you see 20 \* 10, it means multiply 20 by 10. Here's some code for you to try:

**<?php**

**$first\_number = 10;  
$second\_number = 20;  
$sum\_total = $second\_number \* $first\_number;**

**print ($sum\_total);**

**?>**

In the above code, we're just multiplying whatever is inside of our two variables. We're then assigning the answer to the variable on the left of the equals sign. (You can probably guess what the answer is without running the code!)

Just like addition and subtraction, you can multiply more than two numbers:

**<?php**

**$first\_number = 10;  
$second\_number = 20;  
$third\_number = 100;**

**$sum\_total = $third\_number \* $second\_number \* $first\_number;**

**print ($sum\_total);**

**?>**

And you can even do this:

**$sum\_total = $third\_number \* $second\_number \* 10;**

But try this code. See if you can guess what the answer is before trying it out:

**<?php**

**$first\_number = 10;  
$second\_number = 2;  
$third\_number = 3;**

**$sum\_total = $third\_number + $second\_number \* $first\_number;**

**print ($sum\_total);**

**?>**

What answer did you expect? If you were expecting to get an answer of 50 then you really need to know about operator precedence! As was mentioned, some operators (Math symbols) are calculated before others in PHP. Multiplication and division are thought to be more important that addition and division. So these will get calculated first. In our sum above, PHP sees the \* symbol, and then multiplies these two numbers first. When it works out the answer, it will move on to the other symbol, the plus sign. It does this first:

**$second\_number \* $first\_number;**

Then it moves on to the addition. It doesn't do this first:

**$third\_number + $second\_number**

This makes the parentheses more important than ever! Use them to force PHP to work out the sums your way. Here's the two different version. Try them both:

**Version one**  
$sum\_total = $third\_number + **(**$second\_number \* $first\_number**)**;

**Version two**  
$sum\_total = **(**$third\_number + $second\_number**)** \* $first\_number;

Here's we're using parentheses to force two different answers. PHP will work out the sum between the parentheses first, and then move on to the other operator. In version one, we're using parentheses to make sure that PHP does the multiplication first. When it gets the answer to the multiplication, THEN the addition is done. In version two, we're using parentheses to make sure that PHP does the addition first. When it gets the answer to the addition, THEN the multiplication is done.

In the next part, we'll take a look at division.

# PHP Tutorials - Division

To divide one number by another, the**/**symbol is used in PHP. If you see 20 / 10, it means divide 10 into 20. Try it yourself:

**<?php**

**$first\_number = 10;  
$second\_number = 20;  
$sum\_total = $second\_number / $first\_number;**

**print ($sum\_total);**

**?>**

Again, you have to be careful of operator precedence. Try this code:

**<?php**

**$first\_number = 10;  
$second\_number = 20;  
$third\_number = 100;**

**$sum\_total = $third\_number - $second\_number / $first\_number;**

**print ($sum\_total);**

**?>**

PHP won't work out the sum from left to right! Division is done before subtraction. So this will get done first:

**$second\_number / $first\_number**

And NOT this:

**$third\_number - $second\_number**

Using parentheses will clear things up. Here's the two versions for you to try:

**Version one**  
$sum\_total = $third\_number - **(**$second\_number / $first\_number**)**;

**Version two**  
$sum\_total = **(**$third\_number - $second\_number**)** / $first\_number;

The first version will get you an answer of 98, but the second version gets you an answer of 8! So remember this: division and multiplication get done BEFORE subtraction and addition. Use parentheses if you want to force PHP to calculate a different way.

In the next part, we'll take a look at how PHP handles floating point numbers.

# PHP Tutorials - Floating Point Numbers

A floating point number is one that has a dot in it, like 0.5 and 10.8. You don't need any special syntax to set these types of numbers up. Here's an example for you to try:

**<?php**

**$first\_number = 1.2;  
$second\_number = 2.5;  
$sum\_total = $second\_number + $first\_number;**

**print ($sum\_total);**

**?>**

You add up, subtract, divide and multiply these numbers in exactly the same way as the integers you've been using. A warning comes with floating point numbers, though: you shouldn't trust them, if you're after a really, really precise answer!

### Some Exercises

To round up this section on number variables, here's a few exercises (In your print statements, there should be no numbers – just variable names):

**Exercise**  
Write a script to add up the following figures: 198, 134, 76. Use a print statement to output your answer.

**Exercise**  
Write a script to add up the following two numbers: 15, 45. Then subtract the answer from 100. Use a print statement to output your answer.

**Exercise**  
Use variables to calculate the answer to the following sum:

**(200 \* 15) / 10**

Use a print statement to output your answer.

In the next part of these PHP tutorials, we'll take a look at Conditional Logic.

# PHP Tutorials

### If Statements in PHP

You saw in the last section that variables are storage areas for your text and numbers. But the reason you are storing this information is so that you can do something with them. If you have stored a username in a variable, for example, you'll then need to check if this is a valid username. To help you do the checking, something called Conditional Logic comes in very handy indeed. In this section, we'll take a look at just what Conditional Logic is. In the next section, we'll do some practical work.

### Conditional Logic

Conditional Logic is all about asking "What happens IF ... ". When you press a button labelled "Don't Press this Button - Under any circumstance!" you are using Conditional Logic. You are asking, "Well, what happens IF I do press the button?"

You use Conditional Logic in your daily life all the time:

**"If I turn the volume up on my stereo, will the neighbours be pleased?"  
"If spend all my money on a new pair of shoes, will it make me happy?"  
"If I study this course, will it improve my web site?"**

Conditional Logic uses the "IF" word a lot. For the most part, you use Conditional Logic to test what is inside of a variable. You can then makes decisions based on what is inside of the variable. As an example, think about the username again. You might have a variable like this:

**$User\_Name = "My\_Regular\_Visitor";**

The text "**My\_Regular\_Visitor**" will then be stored inside of the variable called**$User\_Name**. You would use some Conditional Logic to test whether or not the variable **$User\_Name** really does contain one of your regular visitors. You want to ask:

**"IF $User\_Name is authentic, then let $User\_Name have access to the site."**

In PHP, you use the "IF" word like this:

**if ($User\_Name = = "authentic") {  
//Code to let user access the site here;  
}**

Without any checking, the if statement looks like this:

**if ( ) {**

**}**

You can see it more clearly, here. To test a variable or condition, you start with the word "if". You then have a pair of round brackets. You also need some more brackets - curly ones. These are just to the right of the letter "P" on your keyboard (Well, a UK keyboard, anyway). You need the left curly bracket first { and then the right curly bracket } at the end of your if statement. Get them the wrong way round, and PHP refuses to work. This will get you an error:

**if ($User\_Name = = "authentic") }  
//Code to Let user access the site here;  
{**

And so will this:

**if ($User\_Name = = "authentic") {  
//Code to Let user access the site here;  
{**

The first one has the curly brackets the wrong way round (should be left then right), while the second one has two left curly brackets.

In between the two round brackets, you type the condition you want to test. In the example above, we're testing to see whether the variable called **$User\_Name** has a value of "authentic":

**($User\_Name = = "authentic")**

Again, you'll get an error if you don't get your round brackets right! So the syntax for the if statement is this:

**if (Condition\_or\_Variable\_to\_test) {  
//your code here;  
}**

In the next lesson, we'll use if statements to display an image on the page.

We'll use the print statement to "print out" HTML code. As an example, take the following HTML code to display an image:

**<IMG SRC =church.jpg>**

Just plain HTML. But you can put that code inside of the print statement:

**print ("<IMG SRC =images/church.jpg>");**

When you run the code, the image should display. Of course, you'll need an image called church.jpg, and in a folder called images.

You can find these amongst the files you can download for this course, in the folder called images. (Go [here](http://www.homeandlearn.co.uk/downloads.html) to get the course files, if you haven't already.)

Copy the images folder to your www (root) directory. Then try the following script:

**<?PHP**

**print ("<IMG SRC =images/church.jpg>");**

**?>**

Save your script to the same folder as the images folder (though NOT inside the images folder). Now fire up your server, and give it a try. Hopefully, you'll see the church image display, as in the following graphic:

[Church Image](http://www.homeandlearn.co.uk/php/php3p1.html) (click to open in a new window 80K)

To clarify things, let's have some more practical example of If Statements.

# PHP Tutorials

### Using If Statements in PHP

We can use an if statement to display our image, from the [previous section](http://www.homeandlearn.co.uk/php/php3p1.html). If the user selected "church", then display the church image. If the user selected "kitten", then display another image (the kitten image, which is also in [your images folder](http://www.homeandlearn.co.uk/downloads.html)). Here's some code:

**<?PHP**

**$kitten\_image = 1;  
$church\_image = 0;**

**if ($kitten\_image == 1) {  
print ("<IMG SRC =images/kitten.jpg>");  
}**

**?>**

Type that out, and save it as **testImages.php**. (Notice how there's no HTML!)

When you run the script, the kitten image should display. Let's look at the code and see what's happening.

The first two lines just set up some variables:

**$kitten\_image = 1;  
$church\_image = 0;**

A value of 1 has been assigned to the variable called **$kitten\_image**. A value of 0 has been assigned to the variable called **$church\_image**. Then we have our if statement. Here it is without the print statement:

**if ($kitten\_image == 1) {**

**}**

Notice how there's no semi-colon at the end of the first line - you don't need one. After the word "if" we have a round bracket. Then comes our variable name:**$kitten\_image**. We want to test what's inside of this variable. Specifically, we want to test if it has a value of 1. So we need the double equals sign (**==**). The double equals sign doesn’t really mean “equals”. It means “has a value of”.

What we want to say is:

**"If the variable called $kitten\_image has a value of 1 then execute some code."**

To complete the first line of the if statement we have another round bracket, and a left curly bracket. Miss any of these out, and you'll probably get the dreaded parse error!  
The code we want to execute, though, is the print statement, so that our kitten image will display. This goes inside of the if statement:

**if ($kitten\_image == 1) {  
print ("<IMG SRC =images/kitten.jpg>");  
}**

You need the semi-colon at the end of the print statement.

But if your if statement only runs to one line, you can just do this:

**if ($kitten\_image == 1) { print ("<IMG SRC = images/kitten.jpg>"); }**

In other words, keep everything on one line. PHP doesn't care about your spaces, so it's perfectly acceptable code. Not very readable, but acceptable!

To make use of the church image, here's some new code to try:

**<?PHP**

**$kitten\_image = 0;  
$church\_image = 1;**

**if ($kitten\_image == 1) {  
print ("<IMG SRC =images/kitten.jpg>");  
}**

**if ($church\_image == 1) {  
print ("<IMG SRC =images/church.jpg>");  
}**

**?>**

Notice that the **$kitten\_image** variable now has a value of 0 and that **$church\_image**is 1. The new if statement is just the same as the first. When you run the script, however, the church image will display. That's because of this line:

**if ($kitten\_image == 1) {**

That says, "If the variable called $kitten\_image has a value of 1 ... ". PHP doesn't bother reading the rest of the if statement, because $kitten\_image has a value of 0. It will jump down to our second if statement and test that:

**if ($church\_image == 1) {**

Since the variable called **$church\_image** does indeed have a value of 1, then the code inside of the if statement gets executed. That code prints out the HTML for the church image:  
print ("<IMG SRC =images/church.jpg>");

In the next section, we'll take a look at**if ... else**statements.

# PHP Tutorials

### if … else Statements in PHP

Instead of using two if statements, as in [the previous lesson](http://www.homeandlearn.co.uk/php/php3p2.html), we can use an **if ... else** statement. Like this:

**<?PHP**

**$kitten\_image = 0;  
$church\_image = 1;**

**if ($kitten\_image == 1) {  
print ("<IMG SRC =images/kitten.jpg>");  
}  
else {  
print ("<IMG SRC =images/church.jpg>");  
}**

**?>**

Copy this new script, save your work, and try it out. You should find that the church image displays in the browser. This time, an **if … else** statement is being used. Let’s see how it works.

The syntax for the if else statement is this:

**if (condition\_to\_test) {**

**}  
else {**

**}**

If you look at it closely, you’ll see that you have a normal If Statement first, followed by an “else” part after it. Here’s the “else” part:

**else {**

**}**

Again, the left and right curly brackets are used. In between the curly brackets, you type the code you want to execute. In our code, we set up two variables:

**$kitten\_image = 0;  
$church\_image = 1;**

The variable called **$kitten\_image** has been assigned a value of 0, and the variable called **$church\_image** has been assigned a value of 1. The first line of the if statement tests to see what is inside of the variable called **$kitten\_image**. It’s testing to see whether this variable has a value of 1.

**if ($kitten\_image == 1) {**

What we’re asking is: “Is it true that $kitten\_image holds a value of 1?” The variable $kitten\_image holds a value of 0, so PHP sees this as not true. Because a value of “not true” has been returned (false, if you like), PHP ignores the line of code for the if statement. Instead, it will execute the code for the “else” part. It doesn’t need to do any testing – else means “when all other options have been exhausted, run the code between the else curly brackets.“ For us, that was this:

**else {  
print ("<IMG SRC =images/church.jpg>");  
}**

So the church image gets displayed. Change your two variables from this:

**$kitten\_image = 0;  
$church\_image = 1;**

To this:

**$kitten\_image = 1;  
$church\_image = 0;**

Run your code again and watch what happens. You should see the kitten! But can you work out why?

In the next section, we'll take a look at **if ... else if** statements.

# PHP Tutorials

### if … else if Statements in PHP

You can also add “else if” parts to the If Statements you've been exploring in [the previous sections](http://www.homeandlearn.co.uk/php/php3p2.html). The syntax is this:

**else if (**another\_condition\_to\_test**) {**

**}**

Change your code to this, to see how else if works:

**<?PHP**

**$kitten\_image = 1;  
$church\_image = 0;**

**if ($kitten\_image == 1) {  
print ("<IMG SRC =images/kitten.jpg>");  
}  
else if ($church\_image == 1){   
print ("<IMG SRC =images/church.jpg>");  
}  
else {   
print ("No value of 1 detected");  
}**

**?>**

Here’s we’re just testing to see which of our variables holds a value of 1. But notice the “else if” lines (and that there’s a space between else and if):

**else if ($church\_image == 1){**  **print ("<IMG SRC =images/church.jpg>");  
}**

What you’re saying is “If the previous if statement isn’t true, then try this one.” PHP will then try to evaluate the new condition. If it’s true (the **$church\_image** variable holds a value of 1), then the code between the new curly brackets gets executes. If it’s false (the**$church\_image** variable does NOT holds a value of 1), then the line of code will be ignored, and PHP will move on.

To catch any other eventualities, we have an “else” part at the end. Notice that all parts (if, else if, and else) are neatly sectioned of with pairs of curly brackets:

**if ($kitten\_image == 1) {**

**}**

**else if ($church\_image == 1) {**

**}**

**else {**

**}**

You can add as many else if parts as you like, one for each condition that you want to test. But change your two variables from this:

**$kitten\_image = 1;  
$church\_image = 0;**

to this:

**$kitten\_image = 0;  
$church\_image = 0;**

Then run your code again. What do you expect to happen?

As a nice example of if statements, there is a file called “**selectPicture.php**” in the files that you [downloaded](http://www.homeandlearn.co.uk/downloads.html). It’s in the **scripts** folder. Copy this to your own www (root) folder. As long as you have all the images mentioned in the script, they should display. But examine the code for the script (ignore the HTML form tags for now). What it does is to display an image, based on what the user selected from a drop down list. If statements are being used to test what is inside of a single variable.

Don’t worry too much about the rest of the code: concentrate on the if statements. All we’re doing is testing what is inside of the variable called **$picture**. We’re then displaying the image that corresponds to the word held in the variable.

Since you will be using if statements a heck of lot in your coding career, it’s essential that you have a good grasp of how to use them. To help you along, there’s some more about Conditional logic in the next section!

# PHP Tutorials

### Comparison Operators in PHP

|  |  |  |
| --- | --- | --- |
| You saw in the last section how to test what is inside of a variable. You used if, else … if, and else. You used the double equals sign (==) to test whether the variable was the same thing as some direct text. The double equals sign is known as a Comparison Operator. There a few more of these “operands” to get used. Here’s a list. Take a look, and then we’ll see a few examples of how to use them. | | |
| **Operand** | **Example** | **Meaning** |
| **==** | $variable1 **==** $variable2 | Has the same value as |
| **!=** | $variable1 **!=** $variable2 | Does NOT have the same value as |
| **<** | $variable1 **<** $variable2 | Less Than |
| **>** | $variable1 **>** $variable2 | Greater Than |
| **<=** | $variable1 **<=** $variable2 | Less than or equals to |
| **>=** | $variable1 **>=** $variable2 | Greater than or equals to |
| Here's some more information on the above Operands.  **= =** (Has the same value as) The double equals sign can mean “Has a value of” or "Has the same value as”. In the example below, the variable called $variable1 is being compared to the variable called $variable2  **if ($variable1 == $variable2) {  }**  **!=** (Does NOT have the same value as) You can also test if one condition is NOT the same as another. In which case, you need the exclamation mark/equals sign combination ( != ). If you were testing for a genuine username, for example, you could say:  **if ($what\_user\_entered != $username) { print("You're not a valid user of this site!")  }**  The above code says, “If what the user entered is NOT the same as the value in the variable called $username then print something out.  **<** (Less Than) You'll want to test if one value is less than another. Use the left angle bracket for this ( < )  **>** (Greater Than) You'll also want to test if one value is greater than another. Use the right angle bracket for this ( > )  **<=** (Less than or equals to) For a little more precision, you can test to see if one variable is less than or equal to another. Use the left angle bracket followed by the equals sign ( <= )  **>=** (Greater than or equals to) If you need to test if one variable is greater than or equal to another, use the right angle bracket followed by the equals sign ( >= )    In the next few sections, you'll see some examples of how to use the comparison operators. You've already used the double equals sign, so we'll start with "**Not equal to**". | | |

# PHP Tutorials

### Not Equal To

In the [previous section](http://www.homeandlearn.co.uk/php/php3p5.html), you saw what Comparison Operators were. In this lessons, we'll explore the Comparison Operator for Not Equal To: **!=.**

So open up your text editor, and add the following script:

**<?PHP'**

**$correct\_username = 'logmein';  
$what\_visitor\_typed = 'logMEin';**

**if ($what\_visitor\_typed != $correct\_username) {  
print("You're not a valid user of this site!");  
}**

**?>**

Save your work and try it out. You should be able to guess what it does! But the thing to note here is the new Comparison Operator. Instead of using the double equals sign we’re now using an exclamation mark and a single equals sign. The rest of the If Statement is exactly the same format as you used earlier.

The things you’re trying to compare need to be different before a value of true is returned by PHP. In the second variable (**$what\_visitor\_typed**), the letters “ME” are in uppercase; in the first variable, they are in lowercase. So the two are not the same. Because we used the NOT equal to operator, the text will get printed. Change your script to this:

**$correct\_username = 'logmein';  
$what\_visitor\_typed = 'logmein';**

**if ($what\_visitor\_typed != $correct\_username) {  
print("You're not a valid user of this site!");  
}  
else {  
print("Welcome back, friend!");  
}**

See if you can figure out what has changed. Before you run the script, what will get printed out?

In the next part, we'll have a look at how to use the Less Than ( < ) and Greater Than ( > ) operators.

# PHP Tutorials

### How to use "Less Than" and "Greater Than" in PHP

The Less Than ( < ) and Greater Than ( > ) symbols come in quite handy. They are really useful in loops (which we'll deal with in another section), and for testing numbers in general.

Suppose you wanted to test if someone has spent more than 100 pounds on your site. If they do, you want to give them a ten percent discount. The Less Than and Greater Than symbols can be used. Try this script. Open up your text editor, and type the following. Save your work, and try it out on your server.

**<?PHP**

**$total\_spent = 110;  
$discount\_total = 100;**

**if ($total\_spent > $discount\_total) {**

**print("10 percent discount applies to this order!");**

**}**

**?>**

By using the great Than symbol ( **>** ), we're saying "If the total spent is greater than the discount total then execute some code."

The Less than symbol can be used in the same way. Change your script to this (new lines are in bold, red text):

**<?PHP**

**$total\_spent = 90;  
$discount\_total = 100;**

**if ($total\_spent > $discount\_total) {  
print("10 percent discount applies to this order!");  
}  
else if($total\_spent < $discount\_total) {  
print("Sorry – No discount!");  
}**

**?>**

In the else if part added above, we're checking to see if the total spent is Less Than ( **<**)100 pounds. If it is, then a new message is display. Notice that the **$total\_spent** variable has been reduced to 90.

There is a problem with scripts such as the ones above, however. In the next part, we'll take a look at the operators for **Less Than or Equal To** and **Greater Than or Equal To**.

# PHP Tutorials

### What these mean: <= >=

We can use the same code you created in [the previous section](http://www.homeandlearn.co.uk/php/php3p7.html) to illustrate "Less Than or Equal To" and "Greater Than or Equal To". Change this line in your code:

**$total\_spent = 90;**

to this:

**$total\_spent = 100;**

Now run your code again. Did anything print?

The reason why nothing printed, and no errors occurred, is because we haven't written any condition logic to test for equality. We're only checking to see if the two variables are either Less Than ( < ) each other, or Greater Than ( > ) each other. We need to check if they are the same (as they now are).

Instead of adding yet another else if part, checking to see if the two totals are equal, we can use the operators **<=** (Less Than or Equal To) or **>=**(Greater Than or Equal To). Here's how. Change this line in your code:

**else if($total\_spent < $discount\_total) {**

to this:

**else if($total\_spent <= $discount\_total) {**

The only thing that's changed is the **Less Than or Equal** to symbol has been used instead of just the **Less Than** sign.

Now run your code again. Because we're now saying "If total spent is Less Than or equal to discount total, then execute the code." So the text gets printed to the screen.

**Exercise**  
Suppose you want to apply the discount if 100 pounds or more has been spent. Change your code above to display the correct message. Use the >= symbol for this exercise.

Comparison Operators can take a little getting used, but are well worth the effort. If you're having a hard time with all these Operands, you'll be glad to hear that there's even more of them! Before we get to them, though, let's take a look at another logic technique you can use – the Switch Statement.

# PHP Tutorials

### The PHP Switch Statement

In some [earlier code](http://www.homeandlearn.co.uk/php/php3p4.html), we tested a single variable that came from a drop-down list. A different picture was displayed on screen, depending on the value inside of the variable. A long list of **if** and **else … if**statements were used. A better option, if you have only one variable to test, is to use something called a switch statement. To see how switch statements work, study the following code:

**<?php**

**$picture ='church';**

**switch ($picture) {  
case 'kitten':  
print('Kitten Picture');  
break;**

**case 'church':  
print('Church Picture');  
break;  
}**

**?>**

In the code above, we place the direct text "church" into the variable called**$picture**. It's this direct text that we want to check. We want to know what is inside of the variable, so that we can display the correct picture.

To test a single variable with a Switch Statement, the following syntax is used:

**switch ($variable\_name) {  
case 'What\_you\_want\_to\_check\_for':  
//code here  
break;  
}**

It looks a bit complex, so we'll break it down.

**switch ($variable\_name) {**  
You Start with the word 'Switch' then a pair of round brackets. Inside of the round brackets, you type the name of the variable you want to check. After the round brackets, you need a left curly bracket.

**case 'What\_you\_want\_to\_check\_for':**  
The word 'case' is used before each value you want to check for. In our code, a list of values was coming from a drop-down list. These value were: church and kitten, among others. These are the values we need after the word 'case'. After the the text or variable you want to check for, a colon is needed ( : ).

**//code here**  
After the semi colon on the 'case' line, you type the code you want to execute. Needless to say, you'll get an error if you miss out any semi-colons at the end of your lines of code!

**break;**  
You need to tell PHP to "Break out" of the switch statement. If you don't, PHP will simply drop down to the next case and check that. Use the word 'break' to get out of the Switch statement.

To see the Switch statement in action, there is a file called "**selectPicture2.php**" amongst the ones you downloaded (Go [here](http://www.homeandlearn.co.uk/downloads.html), if you haven't yet downloaded the files for this course). It’s in the **scripts** folder. Try it out, if you like!

If you look at the last few lines of the Switch Statement in this file, you'll see something else you can add to your own code:

**default:  
print ("No Image Selected");**

The default option is like the **else** from **if … else**. It's used when there could be other, unknown, options. A sort of "catch all" option.

In the next part, we'll take a look at something called Logial Operators.

# PHP Tutorials

|  |  |  |
| --- | --- | --- |
| As well as the PHP comparison operators you saw earlier, there's also something called Logical Operators. You typically use these when you want to test more than one condition at a time. For example, you could check to see whether the username and password are correct from the same If Statement. Here's the table of these Operands. | | |
| **Operand** | **Example** | **Meaning** |
| **&&** | $variable1 **&&** $variable2 | Are both values true? |
| **| |** | $variable1 **| |** $variable2 | Is at least one value true? |
| **AND** | $variable1 **AND** $variable2 | Are both values true? |
| **XOR** | $variable1 **XOR** $variable2 | Is at least one value true, but NOT both? |
| **OR** | $variable1 **OR** $variable2 | Is at least one value true? |
| **!** | **!**$variable1 | Is NOT something |
| The new Operands are rather strange, if you're meeting them for the first time. A couple of them even do the same thing! They are very useful, though, so here's a closer look.  The **&&** Operator The && symbols mean AND. Use this if you need both values to be true, as in our username and password test. After all, you don't want to let people in if they just get the username right but not the password! Here's an example:  **$username ='user'; $password ='password';**  **if ($username =='user' && $password =='password') { print("Welcome back!"); } else { print("Invalid Login Detected"); }**  The if statement is set up the same, but notice that now two conditions are being tested:  **$username =='user' && $password =='password**  This says, "If username is correct AND the password is ok, too, then let them in". Both conditions need to go between the round brackets of your if statement.  The **| |** Operator The two straight lines mean OR. Use this symbol when you only need one of your conditions to be true. For example, suppose you want to grant a discount to people if they have spent more than 100 pounds OR they have a special key. Else they don't get any discount. You'd then code like this:  **$total\_spent =100; $special\_key ='SK12345';**  **if ($total\_spent ==100 | | $special\_key =='SK12345') { print("Discount Granted!"); } else { print("No discount for you!"); }**  This time we're testing two conditions and only need ONE of them to be true. If either one of them is true, then the code gets executed. If they are both false, then PHP will move on.  **AND** and **OR** These are the same as the first two! AND is the same as && and OR is the same as ||. There is a subtle difference, but as a beginner, you can simply replace this:  **$username =='user' && $password =='password**  With this  **$username =='user' AND $password =='password**  And this:  **$total\_spent ==100 | | $special\_key =='SK12345'**  With this:  **$total\_spent ==100 OR $special\_key =='SK12345'**  It's up to you which you use. AND is a lot easier to read than &&. OR is a lot easier to read than ||.  The difference, incidentally, is to do with Operator Precedence. We touched on this when we discussed variables, earlier. Logical Operators have a pecking order, as well. The full table is coming soon!  **XOR** You probably won't need this one too much. But it's used when you want to test if one value of two is true but NOT both. If both values are the same, then PHP sees the expression as false. If they are both different, then the value is true. Suppose you had to pick a winner between two contestants. Only one of them can win. It's an XOR situation!  **$contestant\_one = true; $contestant\_two = true;**  **if ($contestant\_one XOR $contestant\_two) { print("Only one winner!"); } else { print("Both can't win!"); }**  See if you can guess which of the two will print out, before running the script.  The **!**Operator This is known as the NOT operator. You use it test whether something is NOT something else. You can also use it to reverse the value of a true or false value. For example, you want to reset a variable to true, if it's been set to false, and vice versa. Here's some code to try:  **$test\_value = false;**  **if ($test\_value = = false) { print(!$test\_value); }**  The code above will print out the number 1! (You'll see why when we tackle Boolean values below.) What we're saying here is, "If $test\_value is false then set it to what it's NOT." What it's NOT is true, so it will now get this value. A bit confused? It's a tricky one, but it can come in handy!  In the next part, we'll take a look at Boolean values. PHP TutorialsBoolean Values in PHP | | |

A Boolean value is one that is in either of two states. They are known as True or False values, in programming. True is usually given a value of 1, and False is given a value of zero. You set them up just like other variables:

**$true\_value = 1;  
$false\_value = 0;**

You can replace the 1 and 0 with the words "true" and "false" (without the quotes). But a note of caution, if you do. Try this script out, and see what happens:

You can replace the 1 and 0 with the words "true" and "false" (without the quotes). But a note of caution, if you do. Try this script out, and see what happens:

**<?php**

**$true\_value = true;  
$false\_value = false;**

**print ("true\_value = " . $true\_value);  
print (" false\_value = " . $false\_value);**

**?>**

What you should find is that the true\_value will print "1", but the false\_value won't print anything! Now replace true with 1 and false with 0, in the script above, and see what prints out.

Boolean values are very common in programming, and you often see this type of coding:

**$true\_value = true;**

**if ($true\_value) {  
print("that's true");  
}**

This is a shorthand way of saying "if $true\_value holds a Boolean value of 1 then the statement is true". This is the same as:

**if ($true\_value = = 1) {  
print("that's true");  
}**

The NOT operand is also used a lot with this kind of if statement:

**$true\_value = true;**

**if (!$true\_value) {  
print("that's true");  
}  
else {  
print("that's not true");  
}**

You'll probably meet Boolean values a lot, during your programming life. It's worth getting the hang of them!

# PHP Tutorials

### Operator Precedence – a List

|  |  |  |
| --- | --- | --- |
| Here's a list of the operators you've met so far, and the order of precedence. This can make a difference, as we saw during the mathematical operators. Don't worry about these too much, unless you're convinced that your math or logical is correct. In which case, you might have to consult the following: | | |
|  | **\* / %** | **Highest Precedence** |
|  | **+ - .** |  |
|  | **< <= > >=** |  |
|  | **= = =** **!= =** |  |
|  | **&&** |  |
|  | | | |  |
|  | **And** |  |
|  | **XOR** |  |
|  | **OR** | **Lowest Precedence** |
| The only operators you haven't yet met on the list above are the **= = =** and **!= =**operators.  In recent editions of PHP, two new operators have been introduced: the triple equals sign ( = = =) and an exclamation, double equals ( != =). These are used to test if one value has the same as another AND are of the same type. An example would be:  **$number = 3; $text = 'three';**  **if ($number = = = $text) { print("Same"); } else { print("Not the same"); }**  So this asks, "Do the variables match exactly?" Since one is text and the other is a number, the answer is "no", or false. We won't be using these operators much, if at all!    Ok, if all of that has given you a headache, let's move on to some practical work. In the next section, we'll take a look at HTML forms, and how to get data from them. This is so that we can do other things besides printing to the screen. PHP TutorialsThe HTML Form If you know a little HTML, then you know that the FORM tags can be used to interact with your users. Things that can be added to a form are the likes of text boxes, radio buttons, check boxes, drop down lists, text areas, and submit buttons. A basic HTML form with a textbox and a Submit button looks like this:  <html> <head> <title>A BASIC HTML FORM</title> </head> <body>  <FORM NAME ="form1" **METHOD** =" " **ACTION** = "">  <INPUT TYPE = "TEXT" VALUE ="username"> <INPUT TYPE = "Submit" Name = "Submit1" VALUE = "Login">  </FORM> </body> </html>  We won't explain what all the HTML elements do, as this is a book on PHP. Some familiarity with the above is assumed. But we'll discuss the **METHOD**, **ACTION** and **SUBMIT** attributes in the form above, because they are important.  The above form can be found in the [files you download](http://www.homeandlearn.co.uk/downloads.html). It's in the **scripts** folder, and is called **basicForm.php**. Use it as a template, if you like.  So, create the form above. Save your work as **basicForm.php**. (This name will be VERY important!) Start your server, and make sure the form loads ok in your browser. You should be able to see a text box and a Submit button. Here's what it should look like:  A Basic HTML Form  If a user comes to your site and has to login, for example, then you'll need to get the details from textboxes. Once you get the text that the user entered, you then test it against a list of your users (this list is usually stored on a database, which we'll see how to code for in a later section). First, you need to know about the HTML attributes METHOD, ACTION and SUBMIT. We'll explore these in the next few sections. PHP TutorialsThe Method Attribute of HTML Forms If you look at the first line of our form from [the previous page](http://www.homeandlearn.co.uk/php/php4p1.html), you'll notice a **METHOD**attribute:  <FORM NAME ="form1" **METHOD =**" " ACTION = "">  The **Method** attribute is used to tell the browser how the form information should be sent. The two most popular methods you can use are GET and POST. But our METHOD is blank. So change it to this:  <FORM NAME ="form1" **METHOD ="GET"** ACTION = "">  To see what effect using GET has, save your work again and then click the Submit button on your form. You should see this:  The GET Method of a HTML Form  The thing to notice here is the address bar. After **basicForm.php**, we have the following:  **?Submit1=Login**  This is a consequence of using the GET method. The data from the form ends up in the address bar. You'll see a question mark, followed by form data. In the image above,**Submit1** was the NAME of the button, and **Login** was the VALUE of the button (the text on the button). This is what is being returned by the GET method. You use the GET method when the data you want returned is not crucial information that needs protecting.  You can also use POST as the Method, instead of GET. Click below to see the difference. PHP TutorialsThe POST Attribute of HTML Forms In the [previous section](http://www.homeandlearn.co.uk/php/php4p2.html), you saw what happened in the browser's address bar when you used the GET method for Form data. The alternative to GET is to use POST. Change the first line of your FORM to this:  <FORM NAME ="form1" **METHOD ="POST"** ACTION = "">  Close your browser down, and open it back up. Load your basicForm.php page again, and then click the button. Your address bar will then look like this:  The POST Method of a HTML Form  The ?Submit1=Login part from the previous section is now gone! That is because we used POST as the method. Using POST means that the form data won't get appended to the address in the address bar for all to see. We'll use both POST and GET throughout the book. But it depends on the project: if the data is not sensitive then use GET, otherwise use POST.  Another important attribute of the Form tag is Action. Without Action, your forms won't go anywhere! We'll see how this works in the next part. PHP TutorialsThe ACTION Attribute of HTML Forms The Action attribute is crucial. It means, "Where do you want the form sent?". If you miss it out, your form won't get sent anywhere. You can send the form data to another PHP script, the same PHP script, an email address, a CGI script, or any other form of script.  In PHP, a popular technique is to send the script to the same page that the form is on – send it to itself, in other words. We'll use that technique first, but you'll see both techniques in action.  So you need to change the form you have been creating in the [previous sections](http://www.homeandlearn.co.uk/php/php4p1.html), the one that should be called basicForm.php. Locate the following, and amend the ACTION line to this:  <Form Name ="form1" Method ="POST" **ACTION =** "basicForm.php">  So we're going to be sending the form data to exactly the same page as the one we have loaded – to itself. We'll put some PHP on the page to handle the form data. But for now, save your work again and then click your submit button. You won't see anything different, but you shouldn't see any error message either!  Once your script has an Action attribute set, you can then Submit it. Which we'll see in the next part. PHP TutorialsThe Submit Button of a HTML FORM The HTML Submit button is used to submit form data to the script mentioned in the ACTION attribute. Here's ours:  <Form Name ="form1" Method ="POST" **ACTION = "basicForm.php"**>  So the page mentioned in the ACTION attribute is basicForm.php. To Submit this script, you just need a HTML Submit button:  **<INPUT TYPE = "Submit" Name = "Submit1" VALUE = "Login">**  You don't need to do anything special with a Submit button – all the submitting is done behind your back. As long as SUBMIT has an ACTION set, then your data will get sent somewhere. But the NAME attribute of the Submit buttons comes in very handy. You can use this Name to test if the form was really submitted, or if the user just clicked the refresh button. This is important when the PHP script is on the same page as the HTML form. Our Submit button is called "Submit1", but you can call it almost anything you like.    Now that you know about METHOD, ACTION, and SUBMIT, we can move on to processing the data that the user entered. First, how to get values from our text box. PHP TutorialsGetting values from a Text Box with PHP If you've been following along from the [previous sections](http://www.homeandlearn.co.uk/php/php4p1.html) then your **basicForm.php** now has a METHOD and ACTION set. We're going to use these to process text that a user has entered into a text box. The METHOD attribute tells you **how** form data is being sent, and the ACTION attribute tells you **where** it is being sent.  To get at the text that a user entered into a text box, the text box needs a NAME attribute. You then tell PHP the NAME of the textbox you want to work with. Our text box hasn't got a NAME yet, so change [your HTML](http://www.homeandlearn.co.uk/php/php4p1.html) to this:  <INPUT TYPE = "Text" VALUE ="username" **NAME = "username"**>  The NAME of our textbox is "username". It's this name that we will be using in a PHP script.  To return data from a HTML form element, you use the following strange syntax:  **$\_POST['formElement\_name'];**  You can assign this to a variable:  **$Your\_Variable = $\_POST['formElement\_name'];**  Before we explain all the syntax, add the following PHP script to the HTML code you have so far. Make sure to add it the HEAD section of your HTML (the part to add is in bold):  <html> <head> <title>A BASIC HTML FORM</title>  **<?PHP**  **$username = $\_POST['username']; print ($username);**  **?>**  </head>  Save your work again, and click the submit button to run your script. (Don't worry if you see an error message about "Undefined index". Click the button anyway.) You should see this appear above your text box:  Get the Username from the text box  Delete the text "username" from the textbox, and click the button again. Your new text should appear above the textbox. The text box itself, however, will still have "username" in it. This is because the text box is getting reset when the data is returned to the browser. The Value attribute of the text box is what is being displayed.  So how does it work?  The**$\_POST[]** is an inbuilt function you can use to get POST data from a form. If you had METHOD = "GET" on your form, then you'd used this instead:  **$username = $\_GET['username'];**  So you begin with a dollar sign (**$**) and an underscore character ( **\_** ). Next comes the METHOD you want to use, POST or GET. You need to type a pair of square brackets next. In between the square brackets, you type the NAME of your HTML form element –**username**, in our case.  **$\_POST['username'];**  Of course, you need the semi-colon to complete the line.  Whatever the VALUE was for your HTML element is what gets returned. You can then assign this to a variable:  **$username = $\_POST['username'];**  So PHP will look for a HTML form element with the NAME **username**. It then looks at the VALUE attribute for this form element. It returns this value for you to use and manipulate.  At the moment, all we're doing is returning what the user entered and printing it to the page. But we can use a bit of Conditional Logic to test what is inside of the variable. As an example, change your PHP to this:  **$username = $\_POST['username'];**  **if ($username = = "letmein") { print ("Welcome back, friend!"); } else { print ("You're not a member of this site"); }**  We're now checking to see if the user entered the text "letmein". If so, the username is correct; if not, print another message.  Try it out an see what happens. When you first load the page, before you even click the button, you might see the text "You're not a member of this site" displayed above the textbox. That's because we haven't checked to see if the Submit button on the form was clicked.    In the next part, we'll see how to check if the Submit button was clicked. PHP TutorialsChecking if the Submit Button of a HTML Form was Clicked In the [previous section](http://www.homeandlearn.co.uk/php/php4p6.html), you saw how to get text from a textbox when a Submit button on a form was clicked. However, when you first load the page the text still displays.  The reason why the text displays when the page is first loaded is because the script executes whether the button is clicked or not. This is the problem you face when a PHP script is on the same page as the HTML, and is being submitted to itself in the ACTION attribute.  To get round this, you can do a simple check using another IF Statement. What you do is to check if the Submit button was clicked. If it was, then run your code. To check if a submit button was clicked, use this:  **if (isset($\_POST['Submit1'])) { }**  Now that looks a bit messy! But it actually consists of three parts:  **if ( ) { } isset( ) $\_POST['Submit1']**  You know about the if statement. But in between the round brackets, we have **isset( )**. This is an inbuilt function that checks if a variable has been set or not. In between the round brackets, you type what you want isset( ) to check. For us, this is**$\_POST['Submit']**. If the user just refreshed the page, then no value will be set for the Submit button. If the user did click the Submit button, then PHP will automatically return a value. Change you script from the [previous page](http://www.homeandlearn.co.uk/php/php4p6.html) to the following and try it out:  **if (isset($\_POST['Submit1'])) {**  **$username = $\_POST['username'];**  **if ($username = = "letmein") { print ("Welcome back, friend!"); } else { print ("You're not a member of this site"); } }**  The new addition is in bold, red text. Make a note of where all those messy round, square and curly brackets are. Miss one out and you'll get an error!    Int he next part, you'll see how to submit your form data to a PHP script on a different page. PHP TutorialsSetting ACTION to a different PHP Page You don't have to submit your form data to the same PHP page, as we've been doing. You can send it to an entirely different PHP page. To see how it works, try this:  Create the following page, and call it **basicForm2.php**. This is your HTML. Notice the ACTION attribue.  **<html> <head> <title>A BASIC HTML FORM</title> </head> <body>**  **<Form name ="form1" Method ="POST" Action ="submitForm.php">**  **<INPUT TYPE = "TEXT" VALUE ="username" Name ="username"> <INPUT TYPE = "Submit" Name = "Submit1" VALUE = "Login">**  **</FORM> </body> </html>**  Now create the following page, and call it **submitForm.php**.  **<?PHP**  **$username = $\_POST['username'];**  **if ($username = = "letmein") { print ("Welcome back, friend!"); } else { print ("You're not a member of this site"); }**  **?>**  In the PHP script above, notice how there's no HTML tags. And we've left out the code that checks if the Submit button was clicked. That's because there's no PHP left in the first page. The code only gets executed IF the Submit is clicked.  Posting form data to a different PHP script is a way to keep the HTML and PHP separate. But there is a problem with it, which you will have noticed: the script gets executed on a new page. That means your form will disappear!  We'll keep the PHP and HTML together. But there will be times when you do want to send form data to a different PHP page, as we'll see in later sections. PHP TutorialsKeeping the data the user entered In the [previous sections](http://www.homeandlearn.co.uk/php/php4p1.html), you've been following along and building up a HTML form. You've learned how to get the text from a text box on a form, but there is a problem.  When the b**asicForm.php** form is submitted, the details that the user entered get erased. You're left with the VALUE that was set in the HTML. For us, username kept appearing in the text box when the button was clicked. You can keep the data the user entered quite easily.  Your script should now look like the one in the link below. If not copy and paste this script, and test it out on your server. (Save the script as **basicForm.php**.)  [The basicForm.php script](http://www.homeandlearn.co.uk/php/basicFormScript.htm)  If you look at the VALUE attribute of the text box in the HTML from the above script, you'll see that it's set to "**username**". Because the form gets posted back to itself, this value will keep re-appearing in the textbox when the page is submitted. Worse, if you've left the Value attributes empty then everything the user entered will disappear. This can be very annoying, if you're asking the user to try again. Better is to POST back the values that the user entered.  To post the details back to the form, and thus keep the data the user has already typed out, you can use this:  **VALUE="<?PHP print $username ; ?>"**  In other words, the VALUE attribute is now a PHP line of code. The line of code is just this:  **<?PHP print $username ; ?>**  It's a bit hard to read, because it's all on one line.  You also need to amend your PHP code in the HEAD section to include an else statement:  **if (isset($\_POST['Submit1'])) {**  **$username = $\_POST['username'];**  **if ($username = = "letmein") { print ("Welcome back, friend!"); } else { print ("You're not a member of this site"); }** **} else { $username =""; }**  The new addition is in bold, red text. But in the else statement, we're just setting the value of the variable called $username for when the button is NOT clicked, i.e. when the page is refreshed.  However, there are some security issues associated with textboxes (and other form elements). So we'll see a more secure way to handle these in a later section.  But our new line of HTML for our textbox reads like this:  <INPUT TYPE = 'TEXT' Name ='username' **VALUE="<?PHP print $username ; ?>"**>  In other words, we're now printing out the VALUE attribute with PHP code.    Now that you know a few things about getting values from HTML forms, here's a few exercise  **Exercise** Add two text boxes and a Submit button to a HTML form. Invite the user to enter a first name and surname. When the button is clicked, print out the person's full name. Don't worry about what is in the text boxes after the button is clicked.  **Exercise** Using the same form as the previous exercise, display the first name and surname in the textboxes, instead of printing them out.  **Exercise** Suppose your web site has only 5 users. Create a HTML form to check if a visitor is one of the 5 users. Display a suitable message.    In the next section, we'll take a look at how to handle Radion Buttons on a HTML Form. PHP TutorialsPHP and Radio Buttons A Radio Button is a way to restrict users to having only one choice. Examples are : Male/Female, Yes/No, or answers to surveys and quizzes.  Here's a simple from with just two radio buttons and a Submit button:  A Form with Radio Buttons  You can find the code for the page above in the [files you downloaded](http://www.homeandlearn.co.uk/downloads.html), in the scripts folder. The file is called **radioButton.php**. Open it up in your text editor. If you want to copy and paste it, click below.  [The Radio Button Form](http://www.homeandlearn.co.uk/php/radioButtonScript.htm)  Make sure you save your work as radioButton.php, as that's where we're posting the Form – to itself.  To get the value of a radio button with PHP code, again you access the NAME attribute of the HTML form elements. In the HTML above, the NAME of the Radio buttons is the same – "gender". The first Radio Button has a value of "male" and the second Radio Button has a value of female. When you're writing your PHP code, it's these values that are returned. Here's some PHP code. Add it to the HEAD section of your HTML:  **<?PHP $selected\_radio = $\_POST['gender']; print $selected\_radio; ?>**  This is more or less the same code as we used for the text box! The only thing that's changed (apart from the variable name) is the NAME of the HTML form element we want to access – "gender". The last line just prints the value to the page. Again, though, we can add code to detect if the user clicked the Submit button:  **if (isset($\_POST['Submit1'])) { $selected\_radio = $\_POST['gender']; print $selected\_radio; }**  Again, this is the same code you saw earlier – just access the form element called 'Submit1' and see if it is set. The code only executes if it is.  Try out the code. Select a radio button and click Submit button. The choice you made is printed to the page - either "male" or "female". What you will notice, however, when you try out the code is that the dot disappears from your selected radio button after the Submit is clicked. Again, PHP is not retaining the value you selected. The solution for radio Buttons, though, is a little more complex than for text boxes  Radio buttons have another attribute - checked or unchecked. You need to set which button was selected by the user, so you have to write PHP code inside the HTML with these values - checked or unchecked. Here's one way to do it:  **The PHP code:**  **<?PHP**  **$male\_status = 'unchecked'; $female\_status = 'unchecked';**  **if (isset($\_POST['Submit1'])) {**  **$selected\_radio = $\_POST['gender'];**  **if ($selected\_radio = = 'male') { $male\_status = 'checked'; } else if ($selected\_radio = = 'female') { $female\_status = 'checked'; } }**  **?>**  **The HTML FORM code:**  **<FORM name ="form1" method ="post" action ="radioButton.php">**  **<Input type = 'Radio' Name ='gender' value= 'male'  <?PHP print $male\_status; ?> >Male**  **<Input type = 'Radio' Name ='gender' value= 'female'  <?PHP print $female\_status; ?> >Female**  **<P> <Input type = "Submit" Name = "Submit1" VALUE = "Select a Radio Button"> </FORM>**  Did we say a little more complex? OK, it's much more complex than any code you've written so far! Have a look at the PHP code inside the HTML first:  **<?PHP print $female\_status; ?>**  This is just a print statement. What is printed out is the value inside of the variable. What is inside of the variable will be either the word "checked" or the word "unchecked". Which it is depends on the logic from our long PHP at the top of the page. Let's break that down.  First we have two variables at the top of the code:  **$male\_status = 'unchecked'; $female\_status = 'unchecked';**  These both get set to unchecked. That's just in case the page is refreshed, rather than the Submit button being clicked.  Next we have our check to see if Submit is clicked:  **if (isset($\_POST['Submit1'])) {**  **}**  Exactly the same as before. As is the next line that puts which radio button was selected into the variable:  **$selected\_radio = $\_POST['gender'];**  We then need some conditional logic. We need to set a variable to "checked", so we have an **if, else … if**construction:  **if ($selected\_radio = = 'male') {**  **} else if ($selected\_radio = = 'female') {**  **}**  All we're doing is testing what is inside of the variable called $selected\_radio. If it's 'male' do one thing; if it's 'female', do another. But look at what we're doing:  **if ($selected\_radio = = 'male') { $male\_status = 'checked'; } else if ($selected\_radio = = 'female') { $female\_status = 'checked'; }**  If the 'male' button was clicked then set the **$male\_status** variable to a value of 'checked'. If the 'female' option button was clicked then set the **$female\_status** variable to a value of 'checked'.  So the code works because of the values inside of two variables: $male\_status and $female\_status.    Yes, the code is very messy – but radio Buttons can be a tad tricky, when you want to retain the value of the selected item. Speaking of tricky – checkboxes are up next! PHP TutorialsPHP and HTML Checkboxes Like [Radio buttons](http://www.homeandlearn.co.uk/php/php4p10.html), checkboxes are used to give visitors a choice of options. Whereas Radio Buttons restrict users to only one choice, you can select more than one option with Checkboxes.  Here's a page that asks users to choose which course books they want to order:  A Form with Checkboxes  As you can see, five items can be selected. Only three are chosen at the moment. When the button is clicked you, as the programmer, want to do at least two things: record which checkboxes were ticked, and have PHP "remember" which items were chosen, just in case of errors.  You don't want the ticks disappearing from the checkboxes, if the user has failed to enter some other details incorrectly. We saw with Radio Buttons that this can involve some tricky coding. The same is true for checkboxes. Let's have a look at one solution to the problem.  Because the code is a little more complex, we've included it in [the files you downloaded](http://www.homeandlearn.co.uk/downloads.html). The script you're looking for is **checkboxes.php**, and is in the **scripts** folder. Open it up and take a look at the code. Here it is in full, if you want to copy and paste it:  [The Checkboxes Script](http://www.homeandlearn.co.uk/php/checkBoxScript.htm)  Note one thing about the HTML checkbox elements: they all have different NAME values (ch1, ch2 ch3, etc). When we coded for the Radio Buttons, we gave the buttons the same NAME. That's because only one option can be selected with Radio Buttons. Because the user can select more than one option with Checkboxes, it makes sense to give them different NAME values, and treat them as separate entities (but some advocate treating them just like Radio Buttons).  In your PHP code, the technique is to check whether each checkbox element has been checked or not. It's more or less the same as for the radio Buttons. First we set up five variable and set them all the unchecked, just like we did before:  **$ch1 = 'unchecked'; $ch2 = 'unchecked'; $ch3 = 'unchecked'; $ch4 = 'unchecked'; $ch5 = 'unchecked';**  The next thing is the same as well: check to see if the Submit button was clicked:  **if (isset($\_POST['Submit1'])) {**  **}**  Inside of this code, however, we have another **isset( )** function:  **if (isset($\_POST['ch1'])) {**  **}**  This time, we're checking to see if a checkbox was set. We need to do this because of a peculiarity of HTML checkboxes. If they are not ticked, they have no value at all, so nothing is returned! If you try the code without checking if the checkboxes are set, then you'll have to deal with a lot of "Undefined" errors.  If the checkbox is ticked, though, it will return a value. And so the isset( ) function will be true. If the isset( ) function is true, then our code inside of the if statement gets executed:  **if ($ch1 = = 'net') { $ch1 = 'checked'; }**  This is yet another If Statement! But we're just checking the value of a variable. We need to know what is inside of it. This one says, "If the value inside of the variable called **$ch1**is '**net**' then execute some code.  The code we need to execute is to put the text '**checked**' inside of the variable called**$ch1**. The rest of the if statements are the same – one for each checkbox on the form.  The last thing we need to do is to print the value of the variable to the HTML form:  **<Input type = 'Checkbox' Name ='ch1' value ="net"  <?PHP print $ch1; ?> >Visual Basic .NET**  Again, this is the same code you saw with the Radio Buttons. The PHP part is:  **<?PHP print $ch1; ?>**  So we're just printing what is inside of the variable called **$ch1**. This will either be "**unchecked**" or "**checked**",  There are other solution for checkboxes, but none seem simple! The point here, though, is that to get the job done we used Conditional Logic.   How to validate checkboxes using JavaScript Another way to deal with checkboxes, though, is with some JavaScript. The following script was sent to us by Tapan Bhanot. It uses JavaScript to validate the checkboxes before sending it to a PHP script. Note how the checkboxes all have the same name on the HTML form, and that it is being posted to a PHP script called **step2.php**:  [View Tapan's script](http://www.homeandlearn.co.uk/php/checkBoxScript2.htm) (opens in a new window)    You'll learn more about dealing with HTML forms as we go along. For now, we'll leave the subject, and move on. It's a bit of a bumpy ride in the next part, though, as we're tackling loops! PHP TutorialsFor Loops in PHP So what’s a loop then? A loop is something that goes round and round. If I told you to move a finger around in a loop, you’d have no problem with the order (unless you have no fingers!) In programming, it’s exactly the same. Except a programming loop will go round and round until you tell it to stop. You also need to tell the programme two other things - where to start your loop, and what to do after it’s finished one lap (known as the update expression).  You can programme without using loops. But it’s an awful lot easier with them. Consider this.  You want to add up the numbers 1 to 4: 1 + 2 + 3 + 4. You could do it like this  **$answer = 1 + 2 + 3 + 4 print $answer**  Fairly simple, you think. And not much code, either. But what if you wanted to add up a thousand numbers? Are you really going to type them all out like that? It’s an awful lot of typing. A loop would make life a lot simpler. You use them when you want to execute the same code over and over again.  We'll discuss a few flavours of programming loops, but as the For Loop is the most used type of loop, we'll discuss those first.  For Loops Here’s a PHP For Loop in a little script. Type it into new PHP script and save your work. Run your code and test it out.  **<?PHP $counter = 0; $start = 1;**  **for($start; $start < 11; $start++) { $counter = $counter + 1; print $counter . "<BR>";  }**  **?>**  How did you get on? You should have seen the numbers 1 to 10 printed on your browser page.  The format for a For Loop is this:  **for (start value; end value; update expression) {**  **}**  The first thing you need to do is type the name of the loop you’re using, in this case for. In between round brackets, you then type your three conditions:  **Start Value** The first condition is where you tell PHP the initial value of your loop. In other words, start the loop at what number? We used this:  **$start = 1**  We’re assigning a value of 1 to a variable called $start. Like all variables, you can make up your own name. A popular name for the initial variable is the letter i . You can set the initial condition before the loop begins, like we did:  **$start = 1 for($start; $start < 11; $start++) {**  Or you can assign your loop value right in the For Loop code:  **for($start = 1; start < 11; start++) {**  The result is the same – the start number for this loop is 1  **End Value** Next, you have to tell PHP when to end your loop. This can be a number, a Boolean value, a string, etc. Here, we’re telling PHP to keep going round the loop while the value of the variable $start is Less Than 11.  **for($start; $start < 11; $start++) {**  When the value of $start is 11 or higher, PHP will bail out of the loop.  **Update Expression** Loops need a way of getting the next number in a series. If the loop couldn’t update the starting value, it would be stuck on the starting value. If we didn’t update our start value, our loop would get stuck on 1. In other words, you need to tell the loop how it is to go round and round. We used this:  **$start++**  In a lot of programming language (and PHP) the double plus symbol (++) means increment (increase the value by one). It’s just a short way of saying this:  **$start = $start + 1**  You can go down by one (decrement) by using the double minus symbol (--), but we won’t go into that.    So our whole loop reads “Starting at a value of 1, keep going round and round while the start value is less than 11. Increase the starting value by one each time round the loop.”  Every time the loop goes round, the code between our two curly brackets { } gets executed:  **$counter = $counter + 1; print $counter . "<BR>";**  Notice that we’re just incrementing the counter variable by 1 each time round the loop, exactly the same as what we’re doing with the start variable. So we could have put this instead:  **$counter ++**  The effect would be the same. As an experiment, try setting the value of $counter to 11 outside the loop (it’s currently $counter = 0). Then inside the loop, use $counter- - (the double minus sign). Can you guess what will happen? Will it crash, or not? Or will it print something out? Better save your work, just in case!    To get more practice with the For Loop, we'll write a little Times Table programme. PHP TutorialsA PHP "Times Table" Programme In the [previous part](http://www.homeandlearn.co.uk/php/php5p1.html), you saw what a For Loop was. In this section, we'll write a times table programme to illustrate how for loops work.  There's a script called timesTable.php amongst the files you downloaded (in the scripts folder). When loaded into the browser, it looks like this:  There's a script called **timesTable.php** amongst the [files you downloaded](http://www.homeandlearn.co.uk/downloads.html) (in the**scripts** folder.). When loaded into the browser, it looks like this:  Times Table Programme  What we're going to do is to get the values from the textboxes and create a Times Table proramme. When the button is clicked, the output will be something like this:  http://www.homeandlearn.co.uk/php/images/times2.jpg  In other words, when the button is clicked we'll print the Times Table to the page. You can have a different Times Table, depending on what values you enter in the textboxes. To make a start with the coding, move on to the next part. PHP TutorialsThe Code for the PHP Times Table Programme The code for the Times Table in [the previous page](http://www.homeandlearn.co.uk/php/php5p2.html) uses a For Loop. The Start for the loop will come from the Start Number textbox, and the end of the loop will come from the End Number textbox. Here's the code in full (without the HTML):  **<?PHP**  **$times = 2;**  **if (isset($\_POST['Submit1'])) {**  **$start = $\_POST['txtStart']; $end = $\_POST['txtEnd']; $times = $\_POST['txtTimes'];**  **for($start; $start <= $end; $start++) { $answer = $start \* $times; print $start . " multiplied by " . $times . " = " . $answer . "<BR>"; }**  **}**  **?>**  **Code Explanation**  We need all those numbers from the textboxes on [the form](http://www.homeandlearn.co.uk/php/php5p2.html), so we start with:  **$times = 2;**  **if (isset($\_POST['Submit1'])) {**  **$start = $\_POST['txtStart']; $end = $\_POST['txtEnd']; $times = $\_POST['txtTimes'];**  **}**  The first line just puts a value in the variable called $times . This is so that the "Multiply By" textbox will have a default value when the page is loaded.  Next we use the **isset( )** function again, just to check if the user clicked the Submit button. This is exactly the same as you saw in the last section.  To get the values from the textboxes, we use the following:  **$start = $\_POST['txtStart']; $end = $\_POST['txtEnd']; $times = $\_POST['txtTimes'];**  Again, this is code you met in the last section. You just assign the values from the textboxes to the new variables using $\_POST[]. In between the square brackets, we've typed the NAME of the HTML textboxes. So this gives us the values that the user entered on the form. Next comes out For Loop:  **for($start; $start <= $end; $start++) { $answer = $start \* $times; }**  Let's look at that first line again:  **for($start; $start <= $end; $start++) {**  So we have a starting value for our loop, an end value, and an update expression. The starting value is coming from the variable called $start. This will be whatever number the user entered in the first textbox. The default is 1. Look at the end value, though:  **$start <= $end**  The end value is when the value in the variable called $start is less than or equal to the value held in the variable called $end. This works because we're increasing the value of $start each time round the loop. The variable called $end is a fixed value, and comes from the textbox on the form.  The last part of the loop code is the update expression. This tells PHP to increase the value of $start each time round the loop:  **$start++**  The double plus symbol (++) means "add 1 to the number held in $start".  And that's the essence of for loops: provide a start value, an end value, and how you want to update each time round the loop.  The code inside the for loop, however, the code that gets executed each time round the loop, is this:  **$answer = $start \* $times;**  Remember, the variable $times holds the times table, the 2 times table by default. This is being multiplied by whatever is inside the variable $start. Each time round the loop, $start will have a different value – first 1, then 2, then 3, etc. The answer is then stored in the variable that we called $answer. So it's really doing this:  **$answer = 1 \* 2; $answer = 2 \* 2; $answer = 3 \* 2; etc**  Finally, we displayed the result to the page like this:  **print $start . " multiplied by " . $times . " = " . $answer . "<BR>";**  This is just concatenation. See if you can work out what all the parts do!  And that’s it – your very own times table generator. If you have children, show them the programme you wrote. They’ll be very impressed and tell you how brilliant you are. Children are like that.  Of course, your programme is not perfect, which I’m sure the children will discover. Especially if they enter a 10 as the start number and a 1 as the end number. Why doesn't it print anything out? Anything you can do to trap this error? Another if statement somewhere, perhaps? PHP TutorialsWhile Loops in PHP Instead of using a [for loop](http://www.homeandlearn.co.uk/php/php5p1.html), you have the option to use a while loop. The structure of a while loop is more simple than a for loop, because you’re only evaluating the one condition. The loop goes round and round while the condition is true. When the condition is false, the programme breaks out of the while loop. Here’s the syntax for a while loop:  **while (condition) { statement }**  And here’s some code to try. All it does is increment a variable called counter:  $counter = 1; **while ($counter < 11) { print (" counter = " . $counter . "<BR>"); $counter++; }**  The condition to test for is **$counter < 11**. Each time round the while loop, that condition is checked. If counter is less than eleven then the condition is true. When $counter is greater than eleven then the condition is false. A while loop will stop going round and round when a condition is false.  If you use a while loop, be careful that you don’t create an infinite loop. You’d create one of these if you didn’t provide a way for you condition to be evaluated as true. We can create an infinite loop with the while loop above. All we have to do is comment out the line where the $counter variable is incremented. Like this:  $counter = 1; **while ($counter < 11) { print (" counter = " . $counter . "<BR>"); //$counter++; }**  Notice the two forward slashes before **$counter++**. This line will now be ignored. Because the loop is going round and round while counter is less than 11, the loop will never end – $counter will always be 1.  Here’s a while loop that prints out the 2 times table. Try it out in a script.  **$start = 1; $times = 2; $answer = 0;**  **while ($start < 11) { $answer = $start \* $times; print ($start . " times " . $times . " = " . $answer . "<BR>"); $start++; }**  The while loop calculates the 2 times tables, up to a ten times 2. Can you see what’s going on? Make sure you understand the code. If not, it’s a good idea to go back and read this section again. You won’t be considered a failure. Honest!  In the next part, we'll have a brief look at **Do ... While** loops PHP TutorialsDo While loops in PHP This type is loop is almost identical to the [while loop](http://www.homeandlearn.co.uk/php/php5p4.html), except that the condition comes at the end:  **do statement while (condition)**  The difference is that your statement gets executed at least once. In a normal while loop, the condition could be met before your statement gets executed.  Don’t worry too much about do … while loops. Concentrate on [For loops](http://www.homeandlearn.co.uk/php/php5p1.html) and [While loops](http://www.homeandlearn.co.uk/php/php5p4.html). But there is another type of loop that comes in handy - the [For Each loop](http://www.homeandlearn.co.uk/php/php6p5.html). First, a quick word about the **break** statement. PHP TutorialsHow to break out of PHP Loops There are times when you need to break out of a loop before the whole thing gets executed. Or, you want to break out of the loop because of an error your user made. In which case, you can use the break statement. Fortunately, this involves nothing more than typing the word **break**. Here’s some not very useful code that demonstrates the use of the break statement:  **$TeacherInterrupts = true; $counter = 1;**  **while ($counter < 11) { print(" counter = " + $counter + "<BR>"); if ($TeacherInterrupts = = true) break; $counter++; }**  Try the code out and see what happens.  Ok, that's enough of loops. For now. In the next section, we'll take a look at what arrays are, and how useful they can be. (Yes, there'll be loops!) PHP TutorialsWhat is an Array? You know what a variable is – just a storage area where you hold numbers and text. The problem is, a variable will hold only one value. You can store a single number in a variable, or a single string. An array is like a special variable, which can hold more than one number, or more than one string, at a time. If you have a list of items (like a list of customer orders, for example), and you need to do something with them, then it would be quite cumbersome to do this:  **$Order\_Number1 = "Black shoes"; $Order\_Number2 = "Tan shoes"; $Order\_Number3 = "Red shoes"; $Order\_Number4 = "Blue shoes";**  What if you want to loop through your orders and find a specific one? And what if you had not four orders but four hundred? A single variable is clearly not the best programming tool to use here. But an array is! An array can hold all your orders under a single name. And you can access the orders by just referring to the array name.  If that's a bit confusing right now, let’s make a start on explaining how arrays work. PHP TutorialsHow to Set up an Array in PHP In the code on the [previous page](http://www.homeandlearn.co.uk/php/php6p1.html), we had four items, and all with a different variable name: **$Order\_Number1**, **$Order\_Number2**, **$Order\_Number3**, and**$Order\_Number4**. With an array, you can just use a single name. You set up an array like this:  **$Order\_Number = array( );**  First you type out what you want your array to be called (**$Order\_Number**, in the array above) and, after an equals sign, you type this:  **array( );**  So setting up an array just involves typing the word array followed by a pair of round brackets. This is enough to tell PHP that you want to set up the array. But there's nothing in the array yet. All we're doing with our line of code is telling PHP to set up an array, and give it the name $Order\_Number.  You can use two basic methods to put something into an array.  **Method One – Type between the round brackets**  The first method involves typing your values between the round brackets of array(). In the code below, we're setting up an array to hold the seasons of the year:  **$seasons = array("Autumn", "Winter", "Spring", "Summer");**  So the name of the array is $seasons. Between the round brackets of array(), we have typed some values. Each value is separated by a comma:  **("Autumn", "Winter", "Spring", "Summer")**  Arrays work by having a position, and some data for that position. In the above array, "Autumn" is in position zero, "Winter" is in position 1, "Spring" is in position 2, and "Summer" is in position 3.  The first position is always zero, unless you tell PHP otherwise. But the position is know as a Key. The Key then has a value attached to it. You can specify your own numbers for the Keys. If so, you do it like this:  **$seasons = array(1 => "Autumn", 2 => "Winter", 3 => "Spring", 4 => "Summer");**  So you type a number for your key, followed by the equals sign and a right angle bracket ( => ). In the array above, the first Key is now 1 and not 0. The item stored under key 1 is "Autumn". The last key is 4, and the item stored under key 4 is "Summer". Careful of all the commas, when you set up an array like this. Miss one out and you'll get error messages. Here's the keys and values that are set up in the array above:  **1=> "Autumn", 2=> "Winter", 3=> "Spring", 4=> "Summer"**  If you let PHP set the keys for you, it would be this:  **0=> "Autumn",  1=> "Winter", 2=> "Spring",  3=> "Summer"**  You can have numbers for the values of your keys. Here's an array that stores the numbers 10, 20, 30 and 40.  **$Array\_Name = array(10, 20, 30, 40);**  Because no keys were specified, PHP will set your array up like this:  **0=> 10,  1=> 20, 2=> 30,  3=> 40**  Here's the same array again, only this time we're specifying our own key:  **$Array\_Name = array(1 => 10, 2 => 20, 3 => 30, 4 => 40);**  This array will then look like this:  **1=> 10,  2=> 20, 3=> 30,  4=> 40**  So the key name is typed before the => symbol, and the data stored under this key is to the right.  You can store text and numbers in the same array:  **$Array\_Name = array(1 => 10, 2 => "Spring", 3 => 30, 4 => "Summer");**  The above array would then look like this:  **1=> 10,  2=> "Spring", 3=> 30,  4=> "Summer"**  **Method two – Assign values to an array**  Another way to put values into an array is like this:  **$seasons = array();**  **$seasons[]="Autumn"; $seasons[]="Winter"; $seasons[]="Spring"; $seasons[]="Summer";**  Here, the array is first set up with $seasons = array();. This tells PHP that you want to create an array with the name of $seasons. To store values in the array you first type the name of the array, followed by a pair of square brackets:  **$seasons[]**  After the equals sign, you type out what you want to store in this position. Because no numbers were typed in between the square brackets, PHP will assign the number 0 as the first key:  **0=> "Autumn",  1=> "Winter", 2=> "Spring",  3=> "Summer"**  This is exactly the same as the array you saw earlier. If you want different numbers for your keys, then simply type them between the square brackets:  **$seasons[1]="Autumn"; $seasons[2]="Winter"; $seasons[3]="Spring"; $seasons[4]="Summer";**  PHP will then see your array like this:  **1=> "Autumn",  2=> "Winter", 3=> "Spring",  4=> "Summer"**  This method of creating arrays can be very useful for assigning values to an array within a loop. Here's some code:  **$start = 1; $times = 2; $answer = array();**  **for ($start; $start < 11; $start++) { $answer[$start] = $start \* $times; }**  Don't worry if you don't fully understand the code above. The point is that the values in the array called $answer, and the array key numbers, are being assigned inside the loop. When you get some experience with arrays, you'll be creating them just like above!    In the next part, we'll take a look at how to get at the values stored in your arrays. PHP TutorialsGetting at the Values Stored in PHP Arrays OK, so you now know how to [store values in your array](http://www.homeandlearn.co.uk/php/php6p2.html). But how do you get at those values? Well, there are few ways you can do it. But the "Key" is the key. Here's an example for you to try:  **<?php**  **$seasons = array("Autumn", "Winter", "Spring", "Summer"); print $seasons[0];**  **?>**  The array is the same one we set up [before](http://www.homeandlearn.co.uk/php/php6p2.html). To get at what is inside of an array, just type the key number you want to access. In the above code, we're printing out what is held in the 0 position (Key) in the array. You just type the key number between the square brackets of your array name:  **print $Array\_Name[0];**  You can also assign this value to another variable:  **$key\_data = $Array\_Name[0]; print $key\_data;**  It's a lot easier using a loop, though. Suppose you wanted to print out all the values in your array. You could do it like this:  **$seasons = array("Autumn", "Winter", "Spring", "Summer");**  **print $seasons[0]; print $seasons[1]; print $seasons[2]; print $seasons[3];**  Or you could do it like this:  **for ($key\_Number = 0; $key\_Number < 4; $key\_Number++) { print $seasons[$key\_Number]; }**  If you have many array values to access, then using a loop like the one above will save you a lot of work!  You don't have to use numbers for the keys - you can use text. We'll see how to do that in the next part. PHP TutorialsPHP Arrays - Using Text as Keys Your arrays keys don't have to be numbers, as in [the previous section](http://www.homeandlearn.co.uk/php/php6p3.html). They can be text. This can help you remember what's in a key, or what it's supposed to do. When you use text for the keys, you're using an Associative array; when you use numbers for the keys, you're using a Scalar array. Here's an array that sets up first name and surname combinations:  **$full\_name = array( );**  **$full\_name["David"] = "Gilmour"; $full\_name["Nick"] = "Mason"; $full\_name["Roger"] = "Waters"; $full\_name["Richard"] = "Wright";**  Fans of a certain band will know exactly who these people are! But look at the keys and values now:  **David => "Gilmour", Nick => "Mason", Roger => "Waters", Richard => "Wright"**  This is easier to remember than this:  **0 => "Gilmour", 1 => "Mason", 2 => "Waters", 3 => "Wright"**  To access the values in an Associative array, just refer to the Key name:  **print $full\_name["David"];** However, because Associative arrays don't have numbers for the keys, another technique is used to loop round them – the For Each loop. We'll see how they work in the next part.PHP TutorialsPHP Arrays and For Each Loops In the [previous section](http://www.homeandlearn.co.uk/php/php6p4.html), you saw what a Associative array was, and that they use text as the Key. In this lesson, you'll learn how to access each element in Associative array - with the For Each loop. So study the following code (try it out in a script):  **$full\_name = array( );**  **$full\_name["David"] = "Gilmour"; $full\_name["Nick"] = "Mason"; $full\_name["Roger"] = "Waters"; $full\_name["Richard"] = "Wright";**  **foreach ($full\_name as $key\_name => $key\_value) { print "Key = " . $key\_name . " Value = " . $key\_value . "<BR>"; }**  The For Each loop is a little more complex than [other loops](http://www.homeandlearn.co.uk/php/php5p1.html) you've met. In the script above, we set up the array as normal. But the first line of the loop is this:  **foreach ($full\_name as $key\_name => $key\_value) {**  Notice that the name of the loop is one word: foreach and NOT for each. Next comes the round brackets. Inside of the round brackets, we have this:  **$full\_name as $key\_name => $key\_value**  You start by typing the name of the array you want to loop round. For us, that was**$full\_name**. Next is this:  **as $key\_name => $key\_value**  This means, "Get the Key and its Value from the array called $full\_name. The Key is called $key\_name in the script above, and the value is called $key\_value. But these are just variable names. You can call them almost anything you like. Would could have had this:  **foreach ($full\_name as $first\_name => $surname) {**  When you use foreach, PHP knows that it's accessing the key name first and then the key value. It knows this because of the => symbol between the two. It then returns the values into your variable names, whatever they may be.  Once your loop code is executed (a print statement for us), it then loops round and returns the next Key/Value pair, storing the results in your variables.  If you need to access values from an Associative array, then, use a foreach loop.  In the next few sections, you'll see some useful things you can do with arrays. PHP TutorialsSorting Array Values in PHP There may be times when you want to sort the values inside of an array. For example, suppose your array values are not in alphabetical order. Like this one:  **$full\_name = array();**  **$full\_name["Roger"] = "Waters"; $full\_name["Richard"] = "Wright";  $full\_name["Nick"] = "Mason"; $full\_name["David"] = "Gilmour";**  To sort this array, you just use the **assort( )**function. This involves nothing more complex than typing the word asort, followed by round brackets. In between the round brackets, type in the name of your Associative array:  **asort(**$full\_name**);**  The letter "**a**" tells PHP that the array is an Associative one. (If you don't have the "a" before "sort", your key names will turn in to numbers!). The "a" also tells PHP to sort by the Value, and NOT by the key. In our script above, the surnames will be sorted. If you want to sort using the Key, then you can use ksort() instead.  If you have a Scalar array (numbers as Keys), then you leave the "a" off. Like this:  **$numbers = array( );**  **$numbers[]="2"; $numbers[]="8"; $numbers[]="10"; $numbers[]="6";**  **sort($numbers);**  **print $numbers[0] ; print $numbers[1]; print $numbers[2] ; print $numbers[3];**  The numbers are then sorted from lowest to highest. If you want to sort in reverse order then you need the following:  **rsort( )**– Sorts a Scalar array in reverse order **arsort( )**- Sorts the Values in an Associative array in reverse order **krsort( )** - Sorts the Keys in an Associative array in reverse order  In the next part, we look at how to get a random value from an array. PHP TutorialsGet a Random Key from an Array You can grab a random key from an array. This could be useful in games of chance. Here's a simple script that simulates a single dice throw:  **<?PHP**  **$numbers = array(1 => 1, 2 => 2, 3 => 3, 4 => 4, 5 => 5, 6 => 6); $random\_key = array\_rand($numbers, 1); print $random\_key;**  **?>**  The function that returns the random key is this:  **array\_rand(**$numbers, 1**);**  You start off with the function array\_rand( ). In between the round brackets, you need two things: the name of your array, and how many random keys you want to grab.  Try the script out. Refresh the page and you should see a different number between 1 and 6 display.  In the next part, we'll see how the **count** function works when applied to an array. PHP TutorialsThe count function in PHP The **count( )** function is useful when you want to return how many elements are in your array. You can then use this in a for loop. Here's an example we used earlier, only this time with the count function:  **$seasons = array("Autumn", "Winter", "Spring", "Summer");**  **$array\_count = count($seasons);**  **for ($key\_Number = 0; $key\_Number < $array\_count; $key\_Number++) { print $seasons[$key\_Number]; }**  To get how many elements are in the array, we used this:  **$array\_count = count($seasons);**  So you type the word count and then the round brackets. In between the round brackets, you type the name of your array. The function then counts how many elements are in the array, which we then assign to a variable called**$array\_count**. You can then use this value as the end condition in you loop:  **for ($key\_Number = 0; $key\_Number < $array\_count; $key\_Number++)**  Here, we're saying, "keep looping round as long as the value in $key\_Number is less than the value in $array\_count.  To round off this section on arrays, there are some script for you to try out in the next part. PHP TutorialsSome PHP Array Scripts To give your more practice using arrays, there are some scripts for you to try out. The scripts are amongst the [files you downloaded](http://www.homeandlearn.co.uk/downloads.html) (in the scripts folder). The file you're looking for is called scripts.txt. You can also copy and paste the scripts by clicking on the links below.  [**Script One** - Set up an array and print out the values](http://www.homeandlearn.co.uk/php/php6p9s1.html) [**Script Two** - Set up an array with your own Keys](http://www.homeandlearn.co.uk/php/php6p9s2.html) [**Script Three** - Set up an array with mixed values](http://www.homeandlearn.co.uk/php/php6p9s3.html) [**Script four** - Assign values to an array: Method Two example](http://www.homeandlearn.co.uk/php/php6p9s4.html) [**Script Five** - Looping round values in an array](http://www.homeandlearn.co.uk/php/php6p9s5.html) [**Script Six** - Looping round values in an array: example 2](http://www.homeandlearn.co.uk/php/php6p9s6.html) [**Script Seven** - Using text as Keys](http://www.homeandlearn.co.uk/php/php6p9s7.html) [**Script Eight**- Looping round an Associative array using For Each](http://www.homeandlearn.co.uk/php/php6p9s8.html) [**Script Nine** - Sorting Arrays (Associative)](http://www.homeandlearn.co.uk/php/php6p9s9.html) [**Script Ten** - Sorting Arrays (Scalar)](http://www.homeandlearn.co.uk/php/php6p9s10.html)  In the next section, we'll move on to String Manipulation with PHP. PHP TutorialsChanging Case in PHP The ability take strings of text and manipulate them is one of the essential abilities you need as a programmer. If a user enters details on your forms, then you need to check and validate this data. For the most part, this will involve doing things to text. Examples are: converting letters to uppercase or lowercase, checking an email address to see if all the parts are there, checking which browser the user has, trimming white space from around text entered in a text box. All of these come under the heading of string manipulation. To make a start, we'll look at changing the case of character.  Changing the Case of a Character Suppose a you have a textbox on a form that asks users to enter a first name and surname. The chances are high that someone will enter this:  **bill gates**  Instead of this:  **Bill Gates**  So your job as a programmer is to convert the first letter of each name from lower to uppercase. This is quite easy, with PHP.  There's a script amongst the [files you downloaded](http://www.homeandlearn.co.uk/downloads.html) called **changeCase.php**. Open up this page to see the code.  It's just a textbox and a button. The textbox will already have "bill gates" entered, when you load it up. What we want to do is to change it to "Bill Gates" when the button is clicked. Here's the script that does that.  **<?PHP**  **$full\_name = 'bill gates';**  **if (isset($\_POST['Submit1'])) {  $full\_name = $\_POST['username']; $full\_name = ucwords($full\_name); }**  **?>**  The first line just makes sure that the lowercase version is placed into the textbox when the page loads:  **$full\_name = 'bill gates';**  This is the line that we want to convert and turn in to "Bill Gates". The only line in the code that you haven't yet met is this one:  **$full\_name = ucwords($full\_name);**  And that's all you need to convert the first letter of every word to uppercase! The inbuilt function is this:  **ucwords( )**  In between the round brackets, you type the variable or text you want to convert. PHP will take care of the rest. When the conversion is complete, we're storing it back into the variable called $full\_name.  If you just want to convert the first letter of a string (for a sentence, for example), then you can use **ucfirst( )** . Like this:  **$full\_ sentence = ucfirst($full\_ sentence);**  To convert all the letters to either upper or lowercase, use these:  **strtoupper( ) strtolower( )**  Here's an example of how to use them:  **$change\_to\_lowercase = "CHANGE THIS"; $change\_to\_lowercase = strtolower($change\_to\_lowercase);**  **$change\_to\_uppercase = "change this"; $change\_to\_uppercase = strtoupper($change\_to\_lowercase);**  Again, the variable or text you want to change goes between the round brackets of the function. This is then assigned to a variable.    In the next part, we'll take a look at how to trim unwanted white space from your strings. PHP TutorialsTrimming White Space in PHP Another thing you'll want to do is to trim the white (blank) space from text entered into textboxes. This is quite easy, as there's some useful PHP functions to help you do this: Suppose your user has entered this in the textbox:  **" username "**  From the quotation marks, we can see that there is extra space before and after the text. We can count how many characters this string has with another useful function: strlen( ). As its name suggests, this returns the length of a string, By length, we mean how many characters a string has. Try this script:  **<?PHP**  **$space = " username ";**  **$letCount = strlen($space); print $letCount;**  **?>**  When you run the script, you'll find that the variable contains 14 characters. However, username has only 8 characters. If you're checking for an exact match, this matters!  To remove the white space, you can use the **trim( )** function. Change your script to this:  **<?PHP**  **$space = trim(" username ");**  **$letCount = strlen($space); print $letCount;**  **?>**  When you run the script now, you should find that the variable has the correct number of characters - 8. That's because the **trim**( ) function removes any blank spaces from the left and right of a string.  Two related function are **ltrim( )** and **rtrim( )**. The first one, **ltrim( )**, removes space from the beginning of a string; the second one, **rtrim( )**, removes space from the end of a string. You can also use these two functions to trim unwanted characters, as we do much later in the book for the forum walkthrough.  In the next part, we'll take a quick look at the shuffle function. Shuffle characters A rather fun function you can use is **str\_shuffle( )**. What this does is to shuffle all the characters in a string. You can use this to create a quick anagram programme. Try this script:  **<?PHP**  **$full\_name = 'anagram';**  **$full\_name = str\_shuffle($full\_name);**  **print $full\_name;**  **?>**  Only three lines long, but the function takes a variable or direct text and shuffles the characters around.  Next up, something a little more useful - Finding one string inside of another. Finding String Positions with the PHP strpos Function A more useful thing you'll want to do is to see if one string is inside of another. For example, you can get which browser the user has with this:  **$agent =$\_SERVER["HTTP\_USER\_AGENT"]; print $agent;**  If you try it with the Firefox browser, you'd get something like this:  **Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.7.5) Gecko/20041110 Firefox/1.0**  However, Internet Explorer returns something like this:  **Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50215)**  If you're testing which browser the user has, you can use a string function to search for a short string inside of this very long one. A PHP string function you can use is **strpos( )**. The syntax for the **strpos** function is:  **strpos(string\_to\_search, string\_to\_find, start)**  You need to supply at least the first two. The third, start, is optional. Here's a simple example.  **$full\_name = "bill gates"; $letter\_position = strpos($full\_name, "b");  print $letter\_position;**  When you run the script, a value of 0 is returned. That's because PHP considers the first character of the string to be at position 0, the second character at position 1, the third at position 2, etc. Since we were searching for the letter "b", and "bill gates" begins with this letter, a value of 0 is returned.  Try changing **strpos( )** from this:  **$letter\_position = strpos($full\_name, "b");**  to this:  **$letter\_position = strpos($full\_name, "B");**  What happens when you run the script? Nothing! At least, you don't get a value back. That's because if strpos can't find your characters, it returns a value of false. A value of false in PHP can be tested for by using the triple equals operator. Like this.  **$full\_name = "bill gates"; $letter\_position = strpos($full\_name, "B");**  **if ($letter\_position = = = false) { print "Character not found " ; } else { print "Character found"; }**  The triple equals operator ( = = = ) not only checks for a value, remember, but what type of value it is: integer, string, Boolean, etc. If a string is not found, you need to use this operator, just in case the character you're searching for is at position 0. PHP is a little bit quirky with zeros. It seems them as having a false value as well. But it can be a different kind of false! So use = = =.  Here's a script that checks which of four browsers the user has:  **$agent = $\_SERVER['HTTP\_USER\_AGENT'];  if ( strpos(strtoupper($agent), 'MSIE')) {  print "Internet Explorer"; }  else if (strpos(strtoupper($agent), 'FIREFOX')) { print "Firefox"; }  else if (strpos(strtoupper($agent), 'KONQUEROR')) { print "Konqueror"; }  else if (strpos(strtoupper($agent), "LYNX")) { print "Lynx"; }  else {  print $agent; }**  The above script uses two of the string functions that you've met: **strpos( )** and**strtoupper( ).** See if you can figure out what's going on!    In the next part, you'll learn how to split a line of text. You'll need to do this when working with, for example, text files. Splitting a line of text in PHP PHP allows you to split a line of text into its component parts. For example, if you were reading from a text file line by line you might have to break apart a line like this:  **Poll number 1, 1500, 250, 150, 100, 1000**  If this were a poll, and you want to display the results for all to see, then you might be trying to print something like this on the page:  **Poll Number 1** **Respondents: 1500** Answer A: 250 Answer B: 150 Answer C: 100 Answer D: 1000  The line of text is separated by commas. As the line is read in (which we'll see how to do in a later section), you'd be passing it to a variable. You'd then need to chop the text up, based on the comma. We can simulate that. First, pass the text to a variable:  **$text\_line = "Poll number 1, 1500, 250, 150, 100, 1000";**  The next job is to split this text apart, so that PHP knows about all the separate pieces. The pieces we want are:  **Poll number 1** 1500 250 150 100 1000  To split lines of text, the gloriously sounding **explode( )** function can be used. You just provided it with the text you want to split, and the character that is used to separate each piece. Here's the syntax:  **explode(separator, string\_to\_split)**  In between the round brackets of explode( ) the separator you want to use goes first, followed by a comma, then the string you want to split. For our line of code above, you'd do this:  **$text\_line = "Poll number 1, 1500, 250, 150, 100, 1000"; $text\_line = explode("," , $text\_line);**  So we're saying, "Look for a comma in the text, and split the line of text into separate pieces." Once PHP does its job, it puts all the parts into the variable on the left hand side of the equals sign ( = ), which was**$text\_line** for us. This variable will then be an array!  To get at the pieces of the array, access it in the normal manner. Here's some code to try:  **<?PHP**  **$text\_line = "Poll number 1, 1500, 250, 150, 100, 1000"; $text\_line = explode(",",$text\_line);**  **print $text\_line[0];**  **?>**  Run the code and see what happens. Then change the 0 of the print statement to 1, then to 2, then to 3, then to 4, then to 5, and finally to 5. What happens when you enter 6 as the array Key number?  To see all the parts of your array, you can use a different form of print statement. Try changing the print line in your code from this:  **print $text\_line[0];**  to this:  **print\_r($text\_line);**  Run your code and see what happens.  You should see your array details printed out, with all the Keys and the Values. The**print\_r( )**statement is quite useful, when you're trying to debug your code.  And it does show that**explode( )** works – all of the values are in an array!  Another way to access all the element returned by explode( ) is with a for loop:  **$text\_line = "Poll number 1, 1500, 250, 150, 100, 1000"; $text\_line = explode(",",$text\_line);**  **for ($start=0; $start < count($text\_line); $start++) { print $text\_line[$start] . "<BR>"; }**  In the for loop above, we set a start value to zero. The end condition is this:  **$start < count(**$text\_line**)**  We use the count( ) function to get the number of elements in the array called**$text\_line**. Each time round the loop, PHP checks to see if the value in the variable called **$start**is less than how many elements are in the array. It breaks out of the loop when $start is NOT less than count($text\_line).  Inside the loop, we have a normal print statement:  **print $text\_line[$start] . "<BR>";**  To get at each element in the array, this is used:  **$text\_line[$start]**  The variable called **$start** will be different each time round the loop. So the value at each position is printed. The "<BR>" at the end just adds a HTML line break.    As well as exploding a line of text, you'll need to know how to join the parts back together again. We'll do that in the next part. Joining text into a single line In the [previous PHP lesson](http://www.homeandlearn.co.uk/php/php7p5.html), you'll learnt how to split a line of text. All the parts of the line of text will then be in an array.  But if you have a line of text in an array, you can join it all together to form a single line of text. This is just the opposite of explode. This time, use **implode( )**:  **$seasons = array("Autumn", "Winter", "Spring", "Summer"); $new\_textline = implode(",", $seasons);**  Here we have an array called $seasons. The text in the array needs to be joined before writing it back to a text file. The **implode( )** function does the joining. The syntax for the implode( ) function is just the same as [explode( )](http://www.homeandlearn.co.uk/php/php7p5.html).  **implode(separator, text\_to\_join)**  So implode( ) will join all the text together and separate each part with a comma, in the code above. Of course, you don't have to use a comma. You could use a dash:  **$new\_textline = implode("-", $seasons)**  Or any other character:  **$new\_textline = implode("#", $seasons)**  Even a space:  **$new\_textline = implode(" ", $seasons)**  The implode( ) function can come in handy, if you need to work with single lines of text.  In the next part, we'll take a look at something called Escaping. PHP and Escaping Escaping in PHP doesn't mean breaking free and "doing a runner". It is a technique to prevent PHP from ending your strings too early, or for making sure you have the correct string information returned. Here's an example. Try this script:  **<?PHP**  **$string = 'John's Car'; print $string;**  **?>**  Make sure you type the script exactly as it is, with all the single quote marks. Now run the script.  What you should find is that PHP gives you an error message. The reason is that you have three single quote marks. PHP gets confused, because it doesn't know what your string is. To solve the problem, you could use double quotes on the outside. Like this:  **$string = "John's Car";**  Or you could escape the apostrophe. You escape a character by typing a "slash" before it. Like this:  **$string = 'John\'s Car';**  If you try that out, you should find that the string prints correctly.  Now try this script:  **<?PHP**  **$astring = 'mypath\'; print $astring;**  **?>**  Again, you'll get an error from PHP. Surround it with double quotes instead of single quotes and run the script again. Does the string print?  The reason it doesn't is because you haven't escaped the slash. PHP sees it as a special character, and is expecting more details after the slash. But we want a slash in the string. To escape it, use another slash. Like this:  **$astring = 'mypath\\';**  So now we have two slashes on the end of the string. When you run the script, you should find that it prints out this:  **mypath\**  If your PHP script is not returning the characters it should do, then you may need to use the slash to escape them.  You also need to escape certain characters when working with databases, otherwise, you're opening yourself up to attack! You'll hear more on this topic when we get to that section.  In the next part, we'll have a list of string functions that may come in handy, and an example of how to use each one. String function list  |  |  |  | | --- | --- | --- | | Instead of detailing all the possible string functions you can use, we'll just give you a brief list. (There's loads of them!) There's an example of how to use each string function, if you click on the links below. Just dip in to them as and when needed. | | | |  | **String Function** | **Explanation** | |  | [**chr( )**](http://www.homeandlearn.co.uk/php/php7p8s1.html) | Convert an ASCII value to a its equivalent character | |  | [**ord( )**](http://www.homeandlearn.co.uk/php/php7p8s2.html) | Find out what the ASCII value of a character is | |  | [**echo( )**](http://www.homeandlearn.co.uk/php/php7p8s3.html) | An alternative to the print statement | |  | [**similar\_text( )**](http://www.homeandlearn.co.uk/php/php7p8s4.html) | Tells you how similar two strings of text are | |  | [**str\_repeat( )**](http://www.homeandlearn.co.uk/php/php7p8s5.html) | Repeat a character a specified number of times | |  | [**str\_replace( )**](http://www.homeandlearn.co.uk/php/php7p8s6.html) | Replace one string with another | |  | [**str\_word\_count( )**](http://www.homeandlearn.co.uk/php/php7p8s7.html) | Tells you how many words a string has | |  | [**strlen( )**](http://www.homeandlearn.co.uk/php/php7p8s8.html) | Gets the length of a string | |  | [**substr( )**](http://www.homeandlearn.co.uk/php/php7p8s9.html) | Grab a number of character from a string | | Some more examples of string functions can be found on these external sites:  <http://www.w3schools.com/php/func_string_chr.asp> <http://www.zend.com/manual/ref.strings.php>  In the next section, we'll take a look at PHP Functions: what they are, and how to create your own. PHP TutorialsAn Introduction to PHP Functions You've been working with string functions in the [last section](http://www.homeandlearn.co.uk/php/php7p1.html), and references to functions have been made in other sections. But what is a function, and how do you create them? In this section, you'll find out.   What is a Function? A function is just a segment of code, separate from the rest of your code. You separate it because it's nice and handy, and you want to use it not once but over and over. It's a chunk of code that you think is useful, and want to use again. Functions save you from writing the code over and over. Here's an example.  Suppose you need to check text from a textbox. You want to trim any blank spaces from the left and right of the text that the user entered. So if they entered this:  **" Bill Gates "**  You want to turn it into this:  **"Bill Gates"**  But you also want to check if the user entered any text at all. You don't want the textbox to be completely blank!  You can use the PHP inbuilt function called trim( ). Like this:  **$user\_text = trim($\_POST['text1'] );**  That will get rid of the white space in the text box. But it won't check if the text box is blank. You can add an if statement for that:  **if ($user\_text = = "") { error\_message = "Blank textbox detected"; }**  But what if you have lots of textboxes on your form? You'd have to have lots of if statements, and check each single variable for a blank string. That's a lot of code to write!  Rather than do that, you can create a single function, with one if statement that can be used for each blank string you need to check. Using a function means there's less code for you to write. And it's more efficient. We'll see how to write a function for the above scenario in a moment. But first, here's the basic syntax for a function.  **function function\_name( ) {**  **}**  So you start by typing the word function. You then need to come up with a name for your function. You can call almost anything you like. It's just like a variable name. Next, you type two round brackets ( ). Finally, you need the two curly brackets as well { }. Whatever you function does goes between the curly brackets. Here's a simple example that just print something out:  **function display\_error\_message( ) { print "Error Detetceted"; }**  In the example above, we've started with function. We've then called this particular function **display\_error\_message**. In between the curly brackets, there a print statement. Try it out with this script:  **<?PHP function display\_error\_message( ) { print "Error Detetceted"; } ?>**  Run your script and see what happens. You should find that nothing happens!  The reason that nothing happened is because a function is a separate piece of code. It doesn't run until you tell it to. Just loading the script won't work. It's like those inbuilt functions you used, such as **trim**. You can't use trim( ) unless you type out the name, and what you want PHP to trim. The same applies to your own functions – you have to "tell" PHP that you want to use a function that you wrote. You do this by simply typing out the name of your function. This is known as "calling" a function. Try this new version of the script.  **<?PHP**  **function display\_error\_message( ) { print "Error Detetceted"; }**  **display\_error\_message( );**  **?>**  After the function, we've typed out the name again. This is enough to tell PHP to run our code segment. Now change your code to this, and see what happens:  **<?PHP**  **display\_error\_message( );**  **function display\_error\_message( ) { print "Error Detetceted"; }**  **?>**  If you have PHP 4 or above, you should see no difference – the function will still get executed with the name above or below the function. But for neatness and readability's sake, it's better to put all of your function either at the top or bottom of your scripts. Or better yet, in a separate PHP file. You can then use another inbuilt function called "Include" (which we'll get to soon)  In the next part, we'll take a look at something called Scope. PHP TutorialsPHP Variable scope and functions There's a thing called scope in programming. This refers to where in your scripts a variable can be seen. If a variable can bee seen from anywhere, it's said to have global scope. In PHP, variables inside of functions can't be seen from outside of the function. And functions can't see variables if they are not part of the function itself. Try this variation of our script as an example:  **<?PHP**  **$error\_text = "Error Detetceted";**  **display\_error\_message( );**  **function display\_error\_message( ) { print $error\_text; }**  **?>**  This time, we have set up a variable called **$error\_text** to hold the text of our error message. This is set up outside of the function. Run the script, and you'll get a PHP error message about " Undefined variable".  Likewise, try this script:  **<?PHP**  **display\_error\_message( );**  **print $error\_text;**  **function display\_error\_message( ) { $error\_text = "Error message"; }**  **?>**  This time, the variable is inside the function, but we're trying to print it from outside the function. You still get an error message. Here's a correct version:  **<?PHP**  **display\_error\_message( );**  **function display\_error\_message( ) { $error\_text = "Error message"; print $error\_text; }**  **?>**  Here, we have both the variable and the print statement set up inside of the function. The error message now prints.  So if you need to examine what is inside of a variable, you need a way to get the variable to the function. That's where arguments come in. We'll explore arguments in the next part. PHP TutorialsPHP Functions and Arguments In [the previous two sections](http://www.homeandlearn.co.uk/php/php8p1.html), you saw how to set up your own functions, and the importance of Scope. In particular, you learnt that you need a handy way to pass values to your function. That's where arguments come in.   Arguments Functions can be handed variables, so that you can do something with what's inside of them. You pass the variable over to your functions by typing them inside of the round brackets of the function name. Here's a script similar to the one you saw [earlier](http://www.homeandlearn.co.uk/php/php8p2.html):  **<?PHP**  **$error\_text = "Error message"; display\_error\_message($error\_text);**  **function display\_error\_message($error\_text) { print $error\_text; }**  **?>**  The only difference is the that we now have something between the round brackets of our function:  **function display\_error\_message($error\_text) {**  **}**  The name is the same, but we've put a variable in between the round brackets. This is the variable that we want to do something with. The one called **$error\_text.** By typing a variable inside of the round brackets, you are setting up something called an **argument**. An argument is a variable or value that you want your function to deal with.  Notice how the function is called:  **$error\_text = "Error message"; display\_error\_message($error\_text);**  The first line puts something into the variable. But when you want to hand something to a function that has an argument, you need to type it between the round brackets of the function call. In our script, we're typing the name of the variable. But this would do just as well:  **display\_error\_message("Error message");**  Here, we're putting direct text between the round brackets. That works ok. But try it like this:  **$error\_text = "Error message"; display\_error\_message( );**  You'll get an error message from PHP. Something like this:  **"Warning: Missing argument 1 for display\_error\_message( )"**  That's telling you that your function has been set up to take an argument, but that you've left the round brackets empty when you tried to call the function.  Your functions can have more than 1 argument. Just separate each argument with a comma. Like this:  **function error\_check($error\_text, error\_flag) {**  **}**  To call this function, you'd then need to hand it two arguments:  **$error\_text = "Error message"; error\_flag = 1;**  **error\_check($error\_text, error\_flag);**  If you only hand the above function 1 argument, you'd get error messages from PHP.  So, to recap:   * To pass something to a function, create an argument * To call a function that has an argument, don't leave the round brackets empty   In the next part, you'll see a function that checks for blank text boxes. The function has 1 argument. PHP TutorialsA PHP Function to check for blank Textboxes If you remember the script that we wanted to create earlier it was this:   1. Get the text that a user entered in a textbox on a form 2. Trim any blank spaces from the left and right of the text 3. Check that what you have left is not a blank string   So we want to check that the textbox doesn't just contain this "". There has to be something in it, like "Bill Gates". Here's a script that does all three items on our list:  **<?PHP**  **$user\_text = trim("Bill Gates");**  **display\_error\_message($user\_text);**  **function display\_error\_message($user\_text) {**  **if ($user\_text == "") { print "Blank text box detected"; }  else { print "Text OK"; } }**  **?>**  Try it out. When you run the script, you should find that Text OK prints. Now change this line:  **$user\_text = trim("Bill Gates");**  to this:  **$user\_text = trim("");**  Run your script again. This time, Blank text box detected should print out. Obviously, we're not getting the text from a textbox on a form, but just simulating the process. If you want to try out a version with all the HTML, here it is. This next script checks two textboxes on a form.  [A Script to Check for Blank Text Boxes](http://www.homeandlearn.co.uk/php/functionScript.htm)  Try the script out. But the point is, that we're using the same function to check for blank text boxes. We're not writing the same code over and over. Just call our one function as and when needed.  In the next part, we'll see how to get values back out of functions. PHP TutorialsReturning a Value from a PHP Function When you're creating your own functions, you may notice that they can be broken down in to two categories: functions that you can leave, and just let them do their jobs; and functions where you need to get an answer back. As an example, here's the two different categories in action:  **print ("Get on with it!"); $string\_length = strlen($string\_length);**  The print function is an example of a function that you can leave, and just let it do its job. You just tell it what to print and it gets on with it for you. But a function like **strlen( )** is not. You need something back from it – the length of the string.  Suppose you had a function that worked out a 10 percent discount. But you only want to apply the discount if the customer spent over 100 pounds. You could create a function that is handed the amount spent. Then check to see if it's over a 100 pounds. If it is, the function calculates the discount; if not, don't apply the discount. But in both cases, you want the function to return the answer to your question – What do I charge this customer? Here's the script:  **<?PHP**  **$total\_spent = 120; $order\_total = calculate\_total($total\_spent); print $order\_total;**  **function calculate\_total($total\_spent){ $discount = 0.1;**  **if ($total\_spent > 100) { $discount\_total = $total\_spent - ($total\_spent \* $discount); $total\_charged = $discount\_total; } else { $total\_charged = $total\_spent; } return $total\_charged; }**  **?>**  The lines to concentrate on are the coloured lines. The code first sets up a total amount spent, which in practice may come from a form on a text box, or a hidden field:  **$total\_spent = 120;**  The next line is our function call:  **$order\_total = calculate\_total($total\_spent);**  The function call is now on the right of the equals sign ( = ). To the left of the equals sign is just a normal variable - **$order\_total** . If you're setting up your function like this then you are asking PHP to return a value from your functions, and put the answer into a variable on the left of the equals sign. PHP will go off and calculate your function. When it's found an answer, it will try to return a value. The answer will be stored in the name of your function, **calculate\_total( )** for us. But look at the function itself, and the coloured line at the end:  **function calculate\_total($total\_spent) { $discount = 0.1;**  **if ($total\_spent > 100) { $discount\_total = $total\_spent - ($total\_spent \* $discount); $total\_charged = $discount\_total; } else { $total\_charged = $total\_spent; } return $total\_charged; }**  The last line is:  **return $total\_charged;**  The return word tells PHP to return a value. The value it returns is whatever you have stored in the variable that comes after the word return. Here, were telling PHP to set the answer to the function called **calculate\_total( )** to whatever is stored in the variable we've called **$total\_charged**. It's this that will get stored in our variable called**$order\_total**.  If you're finding this a bit tricky, remember what a function is: a separate piece of code that does some work for you. It can either return a value, or not return a value. It depends entirely on your needs. But don't worry about the majority of the code above – just concentrate on the coloured parts.  In the script above, you'd want to get something back from the function, rather than letting it just print something out. If you ran the previous script, you'll notice that the function prints out the same thing twice. To stop that happening, we can get a **return**value, and put it in a variable. We can then check what is coming back from the function, to check what's in it.    In the next part, we'll take a look at some other aspects of functions. PHP TutorialsBy Ref, By Val Functions can be quite hard to get used, if you've never met them before. Another difficult part to understand is how values can change, or not change, depending on scope. Scope, if you recall, refers to where in your code a variable can be seen. If you just do this, for example:  **$Variable\_Value = 10; example( );**  **function example( ) { print $Variable\_Value; }**  then you'll get a PHP error about "undefined variable". That's because the function called example( ) can't see what's inside of the variable called $Variable\_Value.  In order for the function to be able to see what’s inside of the variable called $Variable\_Value, you can set up the function to accept an argument. You'd then type the variable name between the round brackets, when you come to call it. Like this:  **<?PHP**  **$Variable\_Value = 10; example($Variable\_Value);**  **function example($Variable\_Value) { print $Variable\_Value; }**  **?>**  If you run the code above, it now prints out the number ten. But it's important to bear in mind that you are just handing the function a copy of the variable. You're not effecting the original. As an example, change your code to this:  **<?PHP**  **$Variable\_Value = 10;**  **print "Before the function call = " . $Variable\_Value . "<BR>";**  **example($Variable\_Value);**  **print "After the function call = " . $Variable\_Value;**  **function example($Variable\_Value) { $Variable\_Value = $Variable\_Value + 10; print "Inside of the function = " . $Variable\_Value . "<BR>"; }**  **?>**  Here, we have three print statement: one before the call to the function, one inside of the function, and one after the function call. But we're printing out the value of the variable called $Variable\_Value each time. Inside of the function, we're adding ten to the value of the variable. When you run the code, it will print out this:  **Before the function call = 10 Inside of the function = 20 After the function call = 10**  The important one is After the function call. Even though we changed the value of $Variable\_Value inside of the function, it still print 10 after the function call! That's because the function was handed a copy, and NOT the original.  When you hand a function a copy of a variable, it's called passing the variable by value (just a copy). The alternative is to NOT pass a copy, but to refer back to the original. Make one small change to your script. This:  **function example(&$Variable\_Value) {**  The only addition is a & character before the variable between round brackets. This tells PHP that you want to make changes to the original, and don't just want a copy. When you run the script, it now print out the following:  **Before the function call = 10 Inside of the function = 20 After the function call = 20**  After the function call, we now have a value of 20! So a change to the value of the variable outside the function has been made. When you makes changes to the original like this, it's called passing the variable by reference (don't just copy it – remember it).  Try not to worry about value and reference. Unless the answers you're getting back from your function are rather odd, that is!  In the next part, we'll take a look at Server Variables. PHP TutorialsPHP Server Variables PHP stores a list of information about the server. This will include things like, the browser the visitor is using, the IP address, and which web page the visitor came from. Here's a script to try with those three Server Variables.  **$referrer = $\_SERVER['**HTTP\_REFERER**'];** **$browser = $\_SERVER['**HTTP\_USER\_AGENT**'];** **$ipAddress = $\_SERVER['**REMOTE\_ADDR**'];**  **print "Referrer = " . $referrer . "<BR>"; print "Browser = " . $browser . "<BR>"; print "IP Adress = " . $ipAddress;**  These are useful if you want to log your stats, or to ban a particular IP address! (If you run the script on a local machine, you may get an error for the referrer.)  So to get at the values in Server Variables, the syntax is this:  **$\_SERVER[**'Server\_Variable'**]**  You start with a dollar sign, then an underscore character ( $\_ ). Then you add the word**SERVER**. In between square brackets, you type the name of the server variable you want to access. Surround this with either single or double quotes.  Because you are returning a value, you need to put all that on the right hand side of an equals sign. On the left of the equals sign ( = ), you need a variable to hold the string that is returned.  The server variables are held in an array (associative), so you can use a foreach loop to get a list of all available ones. Try this script:  **<?PHP**  **foreach($\_SERVER as $key\_name => $key\_value) { print $key\_name . " = " . $key\_value . "<br>"; }**  **?>**  What the script does is to loop round all the server variables and print out the keys and values in the SERVER array. HTTP Header() Function When you request a web page be brought back to your browser, you're not just bringing back the web page. You're also bringing back something called a HTTP HEADER. This is some extra information, such as type of programme making the request, date requested, should it be displayed as a HTML document, how long the document is, and a lot more besides.  One of the things HTTP HEADER also does is to give status information. This could be whether the page was found (404 errors), and the location of the document. If you want to redirect your users to another page, here's an example:  **<?PHP header("Location: http://www.homeandlearn.co.uk/"); ?>**  **<html> <body> </body> </html>**  Note how the header code goes before any HTML. If you put header code after the HTML, you'll get an error along the lines of "Cannot modify header information." PHP TutorialsThe PHP Include( ) Function Being able to include other files into your HTML code, or for your PHP scripts, is a useful thing. The **include( )** function allows you do this.  Suppose you have a text file that you want to include in a web page that you've already got up and running. You could copy and paste the text from the file straight into you HTML. Or you could use the include( ) function  As an example for you to try, there are two files amongst [the ones you downloaded](http://www.homeandlearn.co.uk/downloads.html) (in the scripts folder), called **include.php** and **textfile.txt**. Load up the one called **include.php**.  Now take a look at the code for this page:  **<HTML> <HEAD> <TITLE>Include files</TITLE> </HEAD>**  **<BODY> <H3>Normal text here </H3> Normal text written in a HTML Editor**  **<H3>Include File here</H3>**  **<?PHP include "textfile.txt" ; ?>**  **</ BODY> </ HTML >**  Our PHP code is in red. Here it is:  **<?PHP include "textfile.txt" ; ?>**  So in between PHP script tags, type the word **include**. After the word include, type the name of the file you want to include on your page. Your filename can either go after a space, and between quotation marks, or you can put it in round brackets (again, with the quotes).  As well as including text, you can include HTML. This can save you lots of work. For example, a web page typically contains a menu bar, with links to other areas of your site. Something like this:  A Menu Bar  Suppose you decide to add a new section to your site. The new page should be like this:  A new section added to the menu bar  If your site contains lots of pages, that would mean having to amend the HTML of all of them. A painful and dreaded task! Instead, use the include( ) function.  To see how it works, load up the page called links.php that is among the files you downloaded (in the scripts folder): you should see the first menu bar. This has the include line, that points to another file - linksPage.txt (this is also in the scripts folder).  If you open up the text file called linksPage.txt, you'll see that it's just a HTML table. To get this table into the PHP page called links.php, we just did this:  **<?PHP include "linksPage.txt" ?>**  The point is, if we had the include line on all pages of out site, and we had to add a new section, we could just change the text file **linksPage.txt**. This change would then mean that all the pages in the site would be updated!  Try it yourself. Add the following line to the page called **linksPage.txt**. Put it between the TABLE tags  **<TR>  <TD height="30" valign="middle" bgcolor="#FFFFCC"> <a href="links.php">New Section</a> </TD> </TR>**  Save the page, and then load up **links.php** again. You should see a new section added to your menu bar.   Including Scripts You can also use the include( ) function for scripts. You could include those valuable error checking functions that you've stored in one PHP file. Or just use it to cut down on the amount of code in the page.  As an example, load up the page called **includeScript.php** (in the scripts folder that[you downloaded](http://www.homeandlearn.co.uk/downloads.html)). The code is quite simple. It's just this:  **<?PHP**  **include "myOtherScript.php";**  **print "This was printed from the includeScript.php"; print "<BR>";**  **doPrint( );**  **?>**  The above script uses include to include another PHP script - myOtherScript.php (also in the scripts folder). The function called doPrint() is in myOtherScript.php. If you open that file, you'll see it's just this:  **<?PHP**  **function doPrint( ) { print "This was printed from the myOtherScript.php"; }**  **?>**  Load up the page called **includeScript.php** in your browser. You should see two lines printed out.  So, **include** is a very useful function – one of the most useful inbuilt PHP functions available to you!    In the next section, we'll look at some of the security issues with the code you've written so far. It's only a short section, but it's essential reading! PHP TutorialsPHP Security Issues and HTML Form Elements If you have things like textboxes and text areas on your forms, then you need to do some security checking on the data that comes in. That's because of things like Cross-Site Scripting. This is when somebody enters scripts into your textboxes to launch an attack on your site. Take this simple form as an example:  **<html> <head> <title>Test Attack</title>**  **<?PHP**  **if ($\_SERVER['REQUEST\_METHOD'] = = 'POST'){ $first\_name = $\_POST['first\_name'];**  **echo $first\_name; }**  **?>**  **</head> <BODY>**  **<Form Method = "Post" action ="testSecurity.php"> <input type = "text" name = "first\_name" value ="test name"> <input type="submit" name="Submit" value="Submit"> </Form>**  **</BODY> </html>**  This form is one of the [files you download](http://www.homeandlearn.co.uk/downloads.html). It can be found in the scripts folder and is called **testSecurity.php**.  Load it up and you'll see that it's just a textbox and a Submit button. Click the button, and you should see "test name" printed on the page.  Now, click inside the textbox and enter the following Javascript:  **<SCRIPT>**alert("Scary Script!")**</SCRIPT>**  Click the Submit button, and then watch what happens. You should see this (you need Javascript enabled in your browser):  Security Form  It's just an alert box. But it could have been something worse!  Another thing someone could do, especially if you have a forum, is to enter HTML directly into your textboxes. They could flood your forum with links to harmful or undesirable web sites. Try this as an example. Delete everything from your textbox, and enter this:  **<A HREF ="nastysite">A Nasty Site</A>**  When you click Submit this time, you should see the following:  Security Form 2  This time, a HTML hyperlink displays above a comments text area. If that was your forum, guess where the link would be?  To stop this kind of thing happening, there are a number of techniques you can use. We'll explore them in the next few parts. PHP TutorialsUsing htmlspecialchars( ) in PHP You can use the inbuilt PHP function htmlspecialchars( ) to convert certain HTML into their respective symbols. (See the [previous lesson](http://www.homeandlearn.co.uk/php/php9p1.html) for why you want to do this.) For example, take the following HTML tag:  **<B>Bold text</B>**  On a web page, that just gives you Bold text. If you enter it into a textbox, and don't convert, then the browser renders it as HTML – in other words, it gives you bold text. The same is true of this:  **<A HREF ="nastysite">A Nasty Site</A>**  This unconverted HTML will turn into an hyperlink. That's because things like left and right pointy brackets are considered to be HTML. The browser sees the code above, and turns it into a hyperlink. It DOESN'T display the left and right pointy brackets. If you actually wanted a left point bracket on your page, you'd use the HTML special character for this symbol:  **&lt;**  And this, essentially, is what the htmlspecialchars( ) function does – turns the HTML into the special character codes.  As an example, change your PHP script from [the previous lesson](http://www.homeandlearn.co.uk/php/php9p1.html) from this,  **$first\_name = $\_POST['first\_name']; echo $first\_name;**  to this:  **$first\_name = $\_POST['first\_name']; $first\_name = htmlspecialchars($first\_name); echo $first\_name;**  Run your code again, and see what happens. You should see this display in the browser:  Consequence of using htmlspecialchars  Now it's not treating the hyperlink as HTML - it's turning it into plain text.  The new line in the script is this:  **$first\_name = htmlspecialchars($first\_name);**  So in between the round brackets of **htmlspecialchars**( ) you type the name of the variable you want to convert to special characters. PHP takes care of the rest.   htmlentities( ) A function similar to htmlspecialchars( ) is **htmlentities**( ). Instead of the above, you can do this:  **$first\_name = $\_POST['first\_name']; $first\_name = htmlentities($first\_name); echo $first\_name;**  The difference between the two is that htmlentities( ) will check for non English language characters, such as French accents, the German umlaut, etc. So if you think your attacker might launch an attack in a language that is not English, then use this.  In the next part, we'll see how to strip HTML tags altogether. PHP TutorialsUsing strip\_tags( ) in PHP A third security option for your HTML forms is to use the strip\_tags( ) function  n. (See the[previous lessons](http://www.homeandlearn.co.uk/php/php9p1.html) for why you want to do this.) It will, as its name suggests, strip all HTML for you. You can, however, tell this function to ignore HTML that you consider harmless, or that you want to include. Here's the syntax:  **strip\_tags($string, html\_tags\_to\_ignore)**  So the first thing you need to provide the **strip\_tags**( ) function with is the string of text you're trying to check. The second thing, **html\_tags\_to\_ignore**, is optional. If you leave this off then the function will strip all tags. Here's two example to try:  **$first\_name = $\_POST['first\_name']; $first\_name = strip\_tags($first\_name); echo $first\_name;**  The new line is set up to strip all HTML from the variable called **$first\_name**. When the script is run, it will look like this:  Using strip_tags  As you can see, only the text of the HTML is left – A Nasty Site.  If it would be OK for people to enter things like bold text or italics, then you'd set up the function like this:  **$first\_name = $\_POST['first\_name']; $first\_name = strip\_tags(**$first\_name**, "<B>"); echo $first\_name;**  So the HTML you want to include goes after a comma, and between quote marks. In the code above, we're allowing the HTML bold tag through. Here's what the text area, and the result looks like:  **Before clicking Submit:**  strip_tags example 2  **After clicking Submit:**  strip_tags example 3  So the HTML in the first picture has been allowed through. In the second picture, you can see that the text is now in bold.    **Summary**  When you have text coming from a form, you should always use a security technique to thwart an attack. However, it's naïve to think we can thwart every attack, and a determined and skilful hacker could probably defeat you. But if you take sensible security measure, you should be able to defend yourself against most attacks. A fuller discussion on security and HTML forms can be found here:  <http://www.secguru.com/param/commonly_asked_cross_site_scripting_questions>    In the next section, we'll take a look at opening and working with files in PHP. PHP TutorialsIntroduction to Working with Files in PHP The ability to open up files such as plain text or CSV files is a great asset for you as a programmer. Not every job requires a database with multiple tables, and storing simple data in a text file can be a good alternative. Especially if your web host doesn't allow you to have a database!  In this section, we take a look at the various way you can manipulate files with PHP. We'll start by opening a plain text file.  For this section, we'll be using the file called **dictionary.txt**, which is amongst [the ones you downloaded](http://www.homeandlearn.co.uk/downloads.html) (in the **scripts** folder).  Open this file up, and take a look at the contents. It should look like this:  **AAS** = Alive and smiling  **ADN** = Any day now  **AEAP** = As early as possible  **AFAIK** = As far as I know  **AFK** = Away from keyboard  **AKA** = Also known as  It's just a long list of values separated by an equals sign. On the left of the equals sign is an abbreviation, and on the right its meaning. We'll open this text file with PHP code. You'll see different ways to open up and work with this text file. Click below to make a start. PHP TutorialsOpening a file with readfile( ) in PHP To open up a file, there are a few methods you can use. The one we'll start with is readfile( ). As it's name suggest, it reads the contents of a file for you. Try this simple script (this assumes that you've read the short introduction on [the previous page](http://www.homeandlearn.co.uk/php/php10p1.html)).  **<?PHP**  **$file\_contents = readfile("dictionary.txt"); print $file\_contents;**  **?>**  Save the script with any file name your like, but make sure it goes in the same folder as your **dictionary.txt** file (see [the previous page](http://www.homeandlearn.co.uk/php/php10p1.html) for an explanation of this file). Run your new code, and see what happens.  You should get a web page full of text, with no separation and no line breaks.  And that's it! Simple, hey? Only two lines of code. You can even get it down to one line:  **print readfile("dictionary.txt");**  But here's the part that does the reading.  **$file\_contents = readfile("dictionary.txt");**  You start by typing readfile, and then a pair of round brackets. In between the round brackets, type the name of the file you want to open. This can be either direct text, as above, or a variable, like this:  **$file\_to\_read = "dictionary.txt"; print readfile($file\_to\_read);**  You don't have to put the file you're trying to read in the same directory. If you had a folder called files in your directory, you could do this:  **$file\_to\_read = "files\dictionary.txt"; print readfile($file\_to\_read);**  Or have any other file reference you want to use.  The **readfile**( ) function is useful if all you want to do is open up a file and read its contents. file\_get\_contents(file\_to\_read); Another function that just reads the contents of a file is **file\_get\_contents**( ). It is available in PHP version 4.3 and above. Here's an example:  **<?PHP**  **$file\_to\_read = "dictionary.txt"; print file\_get\_contents($file\_to\_read);**  **?>**  This used in more or less the same way as the readfile( ) function. The difference for us is the change of name to **file\_get\_contents**( ).  In the next part, we'll take a look at the more commonly used **fopen** function PHP TutorialsOpening a file with fopen( ) in PHP A better method to open files is with **fopen**( ). This function gives you more options that, such as setting whether the file is for reading only, for writing to as well, and a few more options. Here's an example (this assumes that you've read the short introduction on [the previous page](http://www.homeandlearn.co.uk/php/php10p1.html)):  **<?PHP**  **$file\_contents = fopen("dictionary.txt", "r"); print $file\_contents; fclose($file\_contents);**  **?>**  Run this script and see what happens. You should see something like the following printed out:  **Resource id #2**  Not quite what you were expecting! The reason is that **fopen**( ) doesn't actually read the contents of a file. All it does is to set a **pointer** to the file you want to open. It then returns what's call a file handle. All you're doing is telling PHP to remember the location of the file.  The "r" on the end means "open this file for reading only". We'll see [other options](http://www.homeandlearn.co.uk/php/php10p4.html) in a moment. But now that you've told PHP to remember the location of the file you want to open, how do you read the contents of the file?  One way is to use **fgets**( ). This will read a specified number of character on a single line of text. It's typically used to loop round and read each line of text. In the example below, we're printing out each line separately. When you're using **fgets**( ), you also need to check when the end of the file has been reached. This is done with the inbuilt function**feof** - **f**ile, **e**nd **o**f **f**ile. Try the script out, then we'll explain what's happening:  **<?PHP**  **$file\_handle = fopen("dictionary.txt", "r");**  **while (!feof($file\_handle)) {**  **$line\_of\_text = fgets($file\_handle); print $line\_of\_text . "<BR>";**  **}**  **fclose($file\_handle);**  **?>**  What you should find is that the contents are printed out on separate lines. But how does it work?  The first line is this:  **$file\_handle = fopen("dictionary.txt", "r");**  What we're doing here is asking PHP to open up a file, and remember the location. The location is stored as a file handle. We're putting this into a variable called **$file\_handle**. So we haven't yet read the contents of the file – we've just asked PHP to remember where it is.  The next line is tricky! It's a while loop:  **while (!feof($file\_handle)) {**  **}**  There's really two parts to this. There's the while loop:  **while ( ) { }**  And then there's the condition for the while loop:  **!feof($file\_handle)**  A while loop, remember, just goes round and round until you tell it to stop. It goes round and round while a condition is true. The condition between the round brackets was our strange **!feof** line.  The function feof( ) means "file end of file". It tells PHP when the end of a file has been reached. You put the file between the round brackets of the function:  **feof($file\_handle)**  This means, end of the file referred to in the variable called **$file\_handle**. Except, we've used the NOT operator ( ! ):  **!feof($file\_handle)**  That's because we want to keep looping while the end of the has NOT been reached:  **while (!feof($file\_handle)) { }**  This whole line, then, reads: "While the end of the file has NOT been reached, loop round the file pointed to in $file\_handle." As soon as the end of the file has been reached, the while loop will end.  Inside the while loop, the first line is this:  **$line\_of\_text = fgets($file\_handle);**  We're using the fgets( ) function to get a line of text from our file. Again, we need the file handle:  **fgets($file\_handle);**  So we get a line of text from our file, and then place the line into a variable. We then print out the line of text:  **print $line\_of\_text . "<BR>";**  As well as printing out the line of text, we're adding a HTML line break.  The last line in the code is this:  **fclose($file\_handle);**  All this does is to close the open file. It tells PHP that the pointer to the file is no longer needed. You should always close files that you have opened with **fopen**().    The code is a bit tricky, when you're meeting it for the first time. But this kind of file opening is useful when you need to read each line of text. With our file, for example, we could separate each half of the line. We might want to put the abbreviations into one list box and the meanings into another.  Another point to bear in mind about fgets is that it can take (and often does) a second argument – the size of the line to read:  **fgets($file\_handle, line\_size);**  The line size needs to be in bytes. The default is 1024. But this line size is only optional in PHP version 4.2 and above. If your version is earlier than this, then you may get an error if you miss out the line size:  **fgets($file\_handle, 1024);**  If you're really packing a lot of information into each line, then just increase the number for line size.    In the next part, we'll see other options for fopen, rather than the "r" above. Options for fopen( )  |  |  |  | | --- | --- | --- | | In the code from [the previous page](http://www.homeandlearn.co.uk/php/php10p3.html), we used fopen and to get at our file. But we were only reading the file. That's was why we had the letter "r" in the round brackets of fopen( ):  **$file\_handle = fopen("dictionary.txt", "r");**  But there are other options. Here's a fuller list of things you can replace "r" with. | | | |  | **Mode** | **Meaning** | |  | **r** | Use this to read a file only. The pointer is set to the start of the file. | |  | **r+** | Use this to read and write to a file. The pointer is set to the start of the file. | |  | **w** | Use this to write to a file only. It will erase the entire contents of the file you have open. If no file exists with your chosen name, then it will create one for you | |  | **w+** | Same as "w", but used to read and write. | |  | **a** | Use this to write to a file only, and append data to the end of the file. Doesn't erase contents, in other words. | |  | **a+** | Same as "a", but with read access as well. | |  | **x** | Create a file to write only. But gives you a special warning called E\_WARNING. | |  | **x+** | Same as x but with read access as well. | |  | **t** | In Windows, a line break is \r\n. The t converts \n line breaks created on other Operating Systems so that they are readable with Windows | |  | **b** | Force PHP to open the file in binary mode. | | So if you wanted to read and write to the file, you'd use this:  **$file\_handle = fopen("dictionary.txt", "r+");**  Or this, if you want to append data to the end of file when you're writing it back:  **$file\_handle = fopen("dictionary.txt", "a+");**  If you need to work with binary files (like images), then you can add the "b":  **$file\_handle = fopen("dictionary.txt", "rb");**   Checking if the file exists It's a good idea to check if the file exists, before trying to do something with it. The file\_exists( ) function can be used for this:  **if (file\_exists("dictionary2.txt")) { print "file exists"; } else { print "file doesn't exist"; }**  In between the round brackets of file\_exists( ) you then type name of your file. If it does exist, then you can go ahead and do something with it; if not, you can write code to handle any errors.  In the next part, we'll see how to write to a file with PHP. | | |  PHP TutorialsWriting to files in PHP When you need to write to files, there are some more functions you need to use. If you have a version of PHP below version 5, then you can use the **fwrite**() function. But you first need to use [fopen( )](http://www.homeandlearn.co.uk/php/php10p3.html) to get a file handle.  In the next script, we'll try to write some text to a file. We'll use [the "w" option](http://www.homeandlearn.co.uk/php/php10p4.html), as this will create a file for us, if we don't have one with the filename chosen.  **<?PHP**  **$file\_handle = fopen("testFile.txt", "w"); $file\_contents = "Some test text";**  **fwrite($file\_handle, $file\_contents); fclose($file\_handle); print "file created and written to";**  **?>**  The new line is the blue coloured one. First we ask PHP to open the file and create a file handle:  **$file\_handle = fopen("testFile.txt", "w");**  So we're asking PHP to create a file handle that points to a text file called "testFile.txt". If a file of this name can't be found, then one will be created with this name. After a comma, we've typed "w". This tells PHP that the file will be write only.  The third line is where we write to the file:  **fwrite($file\_handle, $file\_contents);**  In between the round brackets of **fwrite**( ), we've placed two things: the file we want to write to, and the contents of the file. And, except for closing the file, that's all you need!  To test to see if it works, run the script. Then look in the folder where you saved the script to. There should now be a file called testFile.txt.  **Exercise** Change the "w" into "a". Run your script a few times, then open the text file. What did you notice?  **Exercise** Change the "a" into "r". Run your script again, then open the text file. What did you notice? Did the contents of the text file change?   file\_put\_contents( ) If you have PHP 5, you can use the new function file\_put\_contents( ) instead of fwrite( ).  It is used in the same way, but has an optional third parameter:  **file\_put\_contents($file\_handle, $file\_contents, context);**  The context option can be **FILE\_USE\_INCLUDE\_PATH**, FILE\_APPEND, **LOCK\_EX.**  So to append to the file, just do this:  **file\_put\_contents($file\_handle, $file\_contents, FILE\_APPEND);**    In the next part, you'll see how to work with CSV files (comma delimited files). PHP TutorialsWorking with CSV files in PHP CSV is a type of file. It means Comma Separated Values. Spreadsheets software like Excel will offer you the opportunity to save files with the CSV extension. We'll see how to work with these types of files now.  The image below shows an Excel spreadsheets. It's just a simple price list:  An Excel price list  The next image shows the spreadsheet being saved as a CSV file:  Save As a CSV file  And here's what the data looks like in a text editor:  A CSV file in a text editor  The above CSV file is the one we want to work with. It shows each line separated by commas.  PHP has a function that allows you to work with CSV file. It's called fgetcsv( ). It's just like the [fgets](http://www.homeandlearn.co.uk/php/php10p3.html)function you used earlier. The difference is that fgetcsv( ) separates each line on the commas, and puts each part into an array. Before trying the next code, make sure you copy the **widgets.csv** file, from the [ones you downloaded,](http://www.homeandlearn.co.uk/downloads.html) over to the same folder where your script is. The widgets.csv file is in the scripts folder. Here's the code:  **<?PHP**  **$file\_handle = fopen("widgets.csv", "r");**  **while (!feof($file\_handle) ) {**  **$line\_of\_text = fgetcsv($file\_handle, 1024);  print $line\_of\_text[0] . $line\_of\_text[1]. $line\_of\_text[2] . "<BR>";**  **}**  **fclose($file\_handle);**  **?>**  When you run the script, you should see all the lines of text printed out. Here's how it works.  The first two lines you've already met: get a handle to the file you want to open, then create a while loop to loop round it.  **$file\_handle = fopen("widgets.csv", "r");**  **while (!feof($file\_handle) ) { }**  Inside the while loop, though, we have our fgetcsv line:  **$line\_of\_text = fgetcsv($file\_handle, 1024);**  In between the round brackets of **fgetcsv**, we've typed to things: our file handle, and the size of the line to read. The size, in this case, is 1024 - 1 kilobyte. You can change this value. When the function gets the line of text, it puts it into the variable we've called**$line\_of\_text**.  But **$line\_of\_text** will now be an array. Each line of our text file looks like this:  **Widget1, blue, £10**  The **fgetcsv** function will split this line when it sees the comma. It then creates a position in the array to hold each part of the line. So it will set up an array with three positions, in our case. Those positions will be as follows:  **$line\_of\_text[0] $line\_of\_text[1] $line\_of\_text[2]**  To print it all out, we had a this line:  **print $line\_of\_text[0] . $line\_of\_text[1]. $line\_of\_text[2] . "<BR>";**  All we're doing here is to print out what is in each position of the array. We've then added a HTML line break, just for display purposes.  If you have CSV files from a spreadsheet that you need to display on a web page, the**fgetcsv** function can come in quite handy!    In the next part, you'll see other options for reading a text file line by line PHP TutorialsReading a text file into an array with PHP There is another option you can use to place lines of text into an array. In the technique below, we're using the [explode( )](http://www.homeandlearn.co.uk/php/php7p5.html) string function to create an array from each line of text. Here's the code (this assumes that you've been following along from [the previous sections](http://www.homeandlearn.co.uk/php/php10p1.html)):  **<?PHP**  **$file\_handle = fopen("dictionary.txt", "rb");**  **while (!feof($file\_handle) ) {**  **$line\_of\_text = fgets($file\_handle); $parts = explode('=', $line\_of\_text);**  **print $parts[0] . $parts[1]. "<BR>"; }**  **fclose($file\_handle);**  **?>**  The lines to examine are in blue; the rest you have met before (get a file handle, loop round, use **fgets** to read the line). The first line to note is this:  **$parts = explode('=', $line\_of\_text);**  If you remember the string section, you'll also be familiar with the explode function. It splits a line of text, based on whatever you have provided for the separator. In our code, we have used the equals sign ( = ) as a separator. This is because each line in the dictionary.txt file looks like this:  **AAS = Alive and smiling**  When the explode function is executed, the variable called **$parts** will be an array. In our text file there will only be two positions in the array, one for each half of the equals sign.  We then print out both parts of the array with this:  **print $parts[0] . $parts[1]. "<BR>";**  So **$parts[0]** will hold the abbreviation (AAS) and **$parts[1]** will hold the meaning.  The next time round the while loop, the second line will be read from the text file. Exactly the same thing happens, so the line will be split again, and placed into an array. This is a good technique to use, if you want to split each line and do something different with each part of the line.    In the next part, we'll take a look at File Locations, and how to work with them. PHP TutorialsFile Locations in PHP There are a few inbuilt PHP functions you can use to find out file paths. This is useful for finding the exact location (relative or absolute) of your scripts or pages. Here's a few example. Before you try these out, create a new PHP page and save it as **fileDir.php**.  **Get the Absolute Path of a File**  **<?PHP**  **$absolute\_path = realpath("fileDir.php");**  **print "Absolute path is: " . $absolute\_path;**  **?>**  To get the exact path of file, then, you can use **realpath**(). In between the round brackets of the function, type the name of the file.  **Get the Directory, but not the file name**  **<?PHP**  **$dir = dirname("folder/myphp/fileDir.php");**  **print "directory is: " . $dir . "<BR>";**  **?>**  To get the names of the folders, you can use the **dirname**( ) function. This will strip off the name of the file and return the rest of the text between the round brackets of the function.  **Get the Filename only**  **<?PHP**  **$bas = basename("folder/myphp/fileDir.php");**  **print "File Name is: " . $bas . "<BR>";**  **?>**  If you only need to get at the name of the file, then use the **basename**( ) function. When you type a longer file path in between the round brackets of the function, it will strip off the rest and leave the name of the file.  **Final Note**  Careful when allowing users to upload files from external sources. This could be used to attack your site. There are a few security implication when working with files. For a more in-depth discussion on this, and other security issues, these links are quite useful:   * <http://www.developer.com/lang/article.php/918141> * <http://www.sklar.com/page/article/owasp-top-ten> * <http://www.devshed.com/c/a/PHP/PHP-Security-Mistakes/> * <http://www.onlamp.com/pub/a/php/2003/07/31/php_foundations.html>   In the next section, we'll move and have a look at how to work with dates and times, PHP TutorialsThe date( ) function in PHP Knowing how to handle date and time values in PHP will be a useful addition to your programming skills. In this and the following sections, we'll take a look at how to process this type of data.   The date( ) function The inbuilt PHP function **date**( ) is the most widely used method of returning date values. Unfortunately, there is a very long list of things you can put between the round brackets of the function! Try this script, to get an idea of how it works:  **<?php**  **$today = date('d-m-y'); print $today;**  **?>**  It should print the day of the week first (d), then the month (m), then the year (y). But this will be the numerical format. So it will print something like:  **04-07-2006**  This type of date can be very confusing, however, because it means the 7th of April in the USA. In the UK, it means the 4th of July.  But to use the function, you first type **date** followed by the round brackets. In between the round brackets you can type a whole host of different date combinations. There's a list coming up. But take note of the case. Change your script to capital letters and watch what happens.  Also, the separator can be anything you like (within reason). So you can have this instead of a hyphen:  **$today = date('d:m:y');**  Or this:  **$today = date('d m y');**  Or even this:  **$today = date('d~m~y');**  Note the single quote marks surrounding the date text. Miss these out and you'll get errors. You can use double quotes, but singles are recommended: dates can be a bit quirky.  Click the next part for a fuller list of the date and time characters to use between the round brackets of **date**. PHP TutorialsUsing the date( ) function in PHP  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | The following is some lists of the date and time characters to use between the round brackets of date. They are all case sensitive. The date function syntax is this, remember:  **date(date\_characters\_here)**  The script from the previous page was this:  **<?php**  **$today = date('d-m-y'); print $today;**  **?>**  The first list is for the day of the week characters. Try them out on your script above.  [**Day of the week Characters**](http://www.homeandlearn.co.uk/php/days.htm) (opens in a new window)  You'll also need the year characters:  [**Year Characters**](http://www.homeandlearn.co.uk/php/year.htm) (opens in a new window)  To add the Time, use the following characters:  [**Time Characters**](http://www.homeandlearn.co.uk/php/time.htm) (opens in a new window)  Here's some other Characters that may come in handy:  [**Other Characters**](http://www.homeandlearn.co.uk/php/other.htm) (opens in a new window)  That's quite a lot of characters! Mostly, you'll be dipping in and out to find the one you need. Here's a few examples of the way you can use the above. Try out the following scripts to see how they work.  **Example 1 (prints out something like Monday 7th September 2006)**  <?PHP  $today = date('l jS F Y'); print $today;  ?>  **Example 2 (prints out something like "It's week 9 of 2006")**  <?PHP  $today = date('W'); $year = date('Y'); print "It's week " . $today . " of " . $year;  ?>  **Example 3 (prints out something like "11:25:44 am")**  <?PHP  $time = date('h:i:s a'); print $time;  ?>  **Example 4 (prints out something like "23:28 GMT Standard Time")**  <?PHP  $time = date('G:i T'); print $time;  ?>    In the next part, we'll see another useful date/time function when we look at **getdate**(). The getdate( ) Function in PHP Another useful date/time function is getdate. This will return an array (associative) with all the date and time values. You can use it for things like comparing one date to another. For example, comparing how many days have passed since a given date. Here's the syntax:  **getdate(time\_stamp);**  The time stamp is optional. If you leave it out, it gets the values for the current local date and time. The parts of the array are this:  **seconds  minutes  hours mday (day of the month as a number) wday (day of the week as a number) mon (month a number) year  yday (year day as a number) weekday (day in text format) month (month in text format) 0 (Seconds since the Unix Epoch)**  Because **getdate** returns an associative array, you can just do this sort of thing:  **$today = getdate();**  **print $today['mday']; print $today['wday']; print $today['yday'];**  So whichever part of the array you want to access goes between square brackets. You then type one of the above Keys between quote marks.  As a further example, suppose you want to work out how many days it's been since a forum member last posted something. And that you have used this to write the date of the last post in a database:  **$post\_date = date('z');**  If you look at the [previous tables](http://www.homeandlearn.co.uk/php/php11p2.html), you'll see that "z" means the year day as a number. So a value of 60 would mean the 60th day of the year.  Now, you've read this value back in, and you want to compare that date against today's date. You can do it like this:  **<?PHP**  **$post\_date = 60; $today = getdate();**  **$day\_difference = $today['yday'] - $post\_date;**  **Print "Days since last post = " . $day\_difference;**  **?>**  So we've set up the array using getdate:  **$today = getdate();**  We've then used "yday" to calculate how many days have elapsed since the last post:  **$day\_difference = $today['yday'] - $post\_date;**  Working with dates and times can be quite tricky, and a good reference is the PHP.net website. As well as setting out all the date and time functions, there's lots of posts from people with good date/time scripts:  <http://uk.php.net/manual/en/function.date.php>    In the next section of the course, we'll explore databases. PHP TutorialsPHP and MySQL: what you need for these tutorials PHP has the ability to connect to and manipulate databases. The most popular database system that is used with PHP is called MySQL. This is a free database system, and comes with the Wampserver software you may have installed at the start of the course. We will be working with MySQL databases throughout these lessons.  We'll create our databases with **phpMyAdmin**, which is part of MySQL. If you installed Wampserver then you can go to the phpMyAdmin section quite easily. Click the icon in the bottom right and select phpMyAdmin from the menu:  http://www.homeandlearn.co.uk/php/images/wampserver_menu2.gif  Another way to bring up phpMyAdmin is to simply type the following address in your browser:  **http://localhost/phpMyAdmin/**  You should see the phpMyAdmin page display:  http://www.homeandlearn.co.uk/php/images/phpMyAdmin.gif  We're going to be creating databases using this. But if you can see the page OK, then you have everything you need. You might want to bookmark the mysql page, though!  If you can't see the page, then either MySQL isn't installed, or it's not configured correctly. Again, you need to refer to the documentation to fix this.  However, we have a few databases amongst [the files you downloaded](http://www.homeandlearn.co.uk/downloads.html). So you can still follow along with the later tutorials.    In the next part, you'll see how to create a database with phpMyAdmin. PHP TutorialsCreating a database using phpMyAdmin  |  |  |  |  |  | | --- | --- | --- | --- | --- | | You can create all of your database tables and queries using PHP code. But before doing that, it's a good idea to get an understanding of just what it is you'll be creating. If you're new to the world of databases, then here's a simple primer.  What is a database and what do they look like? A database is a way to store lots of information. You might want to store the names and addresses of all your contacts, or save usernames and passwords for your online forum. Or maybe customer information.  When you create a database, you're creating a structure like this: | | | | | |  | **ID** | **Title** | **First\_Name** | **Surname** | |  | **1** | Mr | Test | Name | |  | **2** | Mrs | Second | Test | | The columns (ID, Title, First\_Name, Surname) are called **Fields**. The rows are called **Records**. Each record is a separate entry.  In a database, you save the information in a **Table**. A single database can contain many tables, and they can be linked together. When the tables are linked together, it's said to be a **relational** database. If you just have a single table in your database, then it's called a **flat-file** database. Flat-file database are easier to create and understand, so we'll start by creating one of these using phpMyAdmin.  So, if you have version 1.8 of EasyPHP, type **http://127.0.0.1/mysql/** to bring up [phpMyAdmin](http://www.homeandlearn.co.uk/php/php12p1.html). However, if you have EasyPHP version 2.0, type **http://127.0.0.1/home/mysql/** instead.  Although it looks a bit muddled, the part to concentrate on is the textbox under the words **create new database**, as in the next image:  Create a New Database  This is where you type a name for your database. We're going to create a simple Address Book, so type that into the textbox:  Type addressbook into the text box  After you have typed a name for your new database, click the "Create" button. You will be taken to a new area:  The Structure page in phpMyAdmin  In this new area, you can create a Table to go in your database. At the moment, as it says, there are**No tables found in the database**. But the database itself has been created.  To create a new table, type a name for it in the box at the bottom. You can also type a number for the**Fields** textbox. The fields are the columns, remember, and will be things like first\_name, surname, address, etc. You can always add more later, but just type 4 in there. In fact, type it out exactly as it is below:  A new Table in phpMyAdmin  When you've finished, click the **Go** button. Another, more complex, area will appear:  Set up the fields in your Table  In this new area, you set up the **fields** in your database. You can specify whether a field is for text, for numbers, for yes/no values, etc. We'll see how to do that in the next part. PHP TutorialsSetting up Fields in your phpMyAdmin database tables You have four Fields in your table from the [previous section](http://www.homeandlearn.co.uk/php/php12p2.html). Although they are set out in rows in the images, the rows are actually the Columns you saw earlier – the Fields. Each Field needs a name. So go ahead and type the following for your Field names:  Set the Field Names  So we have given each column in our table a name: **ID**, **First\_Name**, **Surname**, and **Address**. The next thing to set is what type of data will be going in to each field - do you want to store text in this field, numbers, Yes/No value, etc?  To set the type of data going into a field, you select an item from the **Type** drop down list. Click the down arrow to see the following list you can choose from:  [**A List of the Field Types**](http://www.homeandlearn.co.uk/php/php12p3.html) (opens in a new window - 21K)  As you can see from the image above, there's quite a lot! But you won't use most them. For the values we have in our four fields, we want to hold these Types:  **ID** – A number, used just to identify each record. This needs to be unique for each record **First\_Name** – Text **Surname** – Text **Address** – Text  If you look at the list, there is an INT but no Number; and there are four different Text Types to choose from. We can use INT (meaning integer) for the numbers, but again, there are a few Integer Types to choose from. And that's leaving out things like float and double. Here's the difference between them, though.  **Integer Values**  **TINYINT** Signed: -128 to 127. Unsigned: 0 to 255  **SMALLINT** Signed: -32768 to 32767. Unsigned: 0 to 65535  **MEDIUMINT** Signed: -8388608 to 8388607. Unsigned: 0 to 16777215  **INT**Signed: -2147483648 to 2147483647. Unsigned: 0 to 4294967295  **BIGINT** Signed: -9223372036854775808. Unsigned: 0 to 18446744073709551615  The signed and unsigned are for minus and non minus values. So if you need to store negative values, you need to be aware of the signed ranges. If you were using a TINYINT value, for example, you can go from minus 128 to positive 127. If you didn't need the minus value, you can go from 0 to positive 255.  For our address book, we have an ID field. We're using this just to identify a record (row). Each record will be unique, so it will need a different number for each. We can set it to one of the INT values. But which one?  If we set ID to TINYINT, then you'd run in to problem if you tried to store more than 255 records. If you used SMALLINT, you'd have problems if you tried to stored the details of friend number 65536. IF you have more than 65 and half thousand friends, then you need a different INT type. We'll assume that you don't, so we'll use SMALLINT.   Text Types The length for the text types can be quite confusing. The MySQL manual says this about the various lengths that each text type can hold:  **TINYTEXT** L+1 byte, where L < 2^8  **TEXT** L+2 bytes, where L < 2^16 **MEDIUMTEXT** L+3 bytes, where L < 2^24 **LONGTEXT** L+4 bytes, where L < 2^32  This in not terribly helpful for beginners! So what does it mean. Well, the L + 1 part means, "The length of the string, plus 1 byte to store the value." The translated values for each are approximately:  **TINYTEXT** 256 bytes **TEXT** 64 KiloBytes **MEDIUMTEXT** 16 MegaBytes **LONGTEXT** 4 GigaBytes  To confuse the issue even more, you can also use CHAR and VARCHAR to store your text. These are quite useful, if you know how many characters you want to store. For example, for a UK postcode you don't need more than 9 characters, and one of those will be a blank space. So there's no sense in setting a postcode field to hold 4 gigabytes! Instead, use CHAR or VARCHAR.  **CHAR** You specify how many characters you want the field to hold. The maximum value is 255. For example:  **CHAR(10)**  This field can then hold a maximum of ten characters. But if you only use 4 of them, the rest of the 10 characters will be blank spaces. The blank spaces get added to the right of your text:  **"TEXT         "**  **"TENLETTERS"**  **VARCHAR** Like CHAR, but the rest of the characters are not padded with blank spaces. The maximum value before MySQL 5.0.3 was 255. After this it's jumped to 65, 535. With VARCHAR, there is also an extra byte that records how long your text is.  For our fields, then, we'll use the following Types:  **ID** SMALLINT **First\_Name** VARCHAR **Surname** VARCHAR **Address** TINYTEXT  So select these from your Types drop down list:  Field Types  We've only set Lengths for the VARCHAR TYPES. If you leave it blank for VARCHAR, you'll get a default value of 1 character.  The other Field settings we'll take a look at are these:  Other Field Settings  **Null** This is an important field in database terminology. It essentially means, "Should the field contain anything?" If you set a field to NOT NULL, then you can't leave it blank when you come to adding records to your database. Otherwise you'll get errors.  **Default** Do you want to add anything to the field, just in case it's left blank when adding a record? If so, type it in here.  **Extra** This is where you can set an auto increment value. This means adding one to the previous record number. This is ideal for us, as we have an ID field. Then we don't have to worry about this field. MySQL will take care of updating it for us.  Field Icons  The three icons are Primary Key, Index, and Unique. Primary keys are not terribly important for flat-file databases like ours. But they are important when you have more than one table, and want to link information. They are set to unique values, like our ID field. An index is useful for sorting information in your tables, as they speed things up. Unique is useful for those fields when there can't be any duplicate values.  So, set a primary key for the ID field by selecting the radio button, and choose Auto Increment from the Extra drop down list:  The auto_increment setting  Your field screen then, minus the parts we've ignored, should look like this:  Field Settings  Bear in mind what we've done here: we've just set up the fields for our table, and specified the kind of information that will be going into each field (the columns). We haven't yet added any information to the table.  Click the Save button on the fields screen. You'll be taken back to the Structure screen. There should be a lot more information there now. Don't worry if it looks a bit confusing. All we want to do is to add one record to the table. We'll then use PHP code to add some more.  In the next part, you'll learn how to add a record to your database table. PHP TutorialsAdding Records to a Table in phpMyAdmin To insert a new record to the table you created in the [previous section](http://www.homeandlearn.co.uk/php/php12p3.html), select the **Insert** link at the top of the page:  Click on Insert  When you click on Insert, you'll be taken to a new area. This one:  The Insert page  As you can see, our four fields are there: ID, First\_Name, Surname, and Address. But look at the lengths of the textboxes under the **Value**. The sizes are determined by the length of the Fields. The address area is a lot bigger, because we used TINYTEXT.  To enter a new record in your table, you type your data in the textboxes under the Value heading. Go ahead and enter the following information for the **Value** textboxes:  **ID**: 1 **First\_Name**: Test **Surname**: Name **Address**: 12 Test Street  Your screen should then look like this:  Values entered into the fields  Finally, click the Go button at the bottom of the screen to create the Row in your table. You will be returned to the Structure screen.  And that's it - you now have a database to work with. To see where it has been saved, navigate to your Wamp folder on your hard drive. Double click the folder called **bin\mysql\mysqlx.x.x**. Inside this folder will be one called **data**. This is where all of your databases are stored:  http://www.homeandlearn.co.uk/php/images/database_dir.gif  Notice the folder name in the image above: testDB. This is the same as the database name, and is automatically created for you for all new databases. When you double click this folder, you should see a few files there:  http://www.homeandlearn.co.uk/php/images/database_dir_tables.gif  Notice the files names - they are the same as the tables you create. In other words, they ARE the tables.  If you have PHP web space, you can upload this folder and its contents to your **data** folder, and you should then be able to access the tables in the database with PHP code.  We can move on to doing just that - accessing this database with some PHP code.  **PHP Tutorials** How to access a MySQL Database with PHP code In this section, you'll see how to manipulate the simple Address Book [database you've just created](http://www.homeandlearn.co.uk/php/php12p1.html). Using PHP code, you'll first open the database. Once the database is open, you can then read its contents. You'll also need to know how to add new records, and delete records. First, though, a database has to be opened, before you can do anything with it.   Opening a Connection to a MySQL database PHP has a lot of inbuilt functions you can use to manipulate databases. In PHP version 5, a lot more were added as well! Here, we'll stay with the inbuilt functions for versions earlier than PHP 5. But if you have version 5, it's well worth researching the newer database functions. A good place to start is php.net. To open [our Address Book database](http://www.homeandlearn.co.uk/php/php12p2.html), we'll use the following inbuilt functions:  **mysql\_connect( ) mysql\_select\_db() mysql\_close()**  The approached we'll take has three steps:   1. Open a connection to MySQL itself 2. Specify the database we want to open 3. Close the connection   Let's do Step 1 on the list.   Step 1 - Open a connection to MySQL The first job is to actually connect to MySQL. As it's name suggests, **mysql\_connect( )**does exactly that. Here's the code we're going to be using. But this is just to get your started. It is recommended that you don't dash off and use this on the internet! This is for learning purposes only.  **<?PHP**  **$user\_name = "root"; $password = ""; $database = "addressbook"; $server = "127.0.0.1";**  **mysql\_connect($server, $user\_name, $password);  print "Connection to the Server opened";**  **?>**  Save your work and try it out on your server (this assumes that you have the Address Book database [we created earlier](http://www.homeandlearn.co.uk/php/php12p2.html), and that it is in the [data folder of MySQL](http://www.homeandlearn.co.uk/php/php12p4.html). If you don't, you can download all the files [here](http://www.homeandlearn.co.uk/downloads.html).).  The first four lines are just setting up variables, and putting something in them:  **$user\_name = "root"; $password = ""; $database = "addressbook"; $server = "127.0.0.1";**  The username we're trying here is "root" and the password is blank. These are the MySQL defaults. You don't need to change these, in most cases.  Hopefully, you won't have any errors. But the line that connects to MySQL is this:  **mysql\_connect($server, $user\_name, $password);**  So you type the name of the function first (**mysql\_connect**), followed by the round brackets. In between the round brackets, you need three things: the name of your server, your MySQL username, and your MySQL password. These can be entered directly, like this:  **mysql\_connect('127.0.0.1', 'root', '');**  Or as variables, like we did at first:  **$user\_name = "root"; $password = ""; $server = "127.0.0.1";**  **mysql\_connect($server, $user\_name, $password);**  And that's all you need to get you connected to MySQL. But we haven't connected to the database yet. That's Step 2 on our list. Step 2 - Specify the database you want to open In our code, we set up a variable with the name of our database:  **$database = "addressbook";**  We now need to do something with this variable. So add this new line to your code (the new line is in red text):  **$user\_name = "root"; $password = ""; $database = "addressbook"; $server = "127.0.0.1";**  **mysql\_connect($server, $user\_name, $password); $db\_found = mysql\_select\_db($database);  print "Connection to the Server opened";**  You use the **mysql\_select\_db**( ) function to specify which database you want to open. The function then returns a true/false value. If it finds your database, a value of **true** is returned; if your database can't be found then a value of **false** is returned. You can use some logic to test if the database was found. Change the last two lines of your code to this:  **$db\_found = mysql\_select\_db($database);**  **if ($db\_found) { print "Database Found"; } else { print "Database NOT Found"; }**  Now change the database name from this:  **$database = "addressbook";**  to something like this:  **$database = "addressbook2";**  Run your code again, and you should see Database NOT Found printed out (unless you have a database called addressbook2). Change the database name back to **addressbook**.  But there's another option you can use for **mysql\_select\_db** - something called a resource link identifier. It's just a file handle that you used in an [earlier section](http://www.homeandlearn.co.uk/php/php10p3.html) (opening text files). You use it like this:  **$user\_name = "root"; $password = ""; $database = "addressbook"; $server = "127.0.0.1";**  **$db\_handle = mysql\_connect($server, $user\_name, $password);**  **$db\_found = mysql\_select\_db($database, $db\_handle);**  **if ($db\_found) { print "Database Found " . $db\_handle; } else { print "Database NOT Found " . $db\_handle; }**  So when we connect to the database, we're now using this:  **$db\_handle = mysql\_connect($server, $user\_name, $password);**  It's just the same as before, except we're returning a value from the **mysql\_connect** function, and putting it into a variable called **$db\_handle**. When we connect to the database, we can use this file handle:  **$db\_found = mysql\_select\_db($database, $db\_handle);**  The resource link identifier (file handle) goes after the name of the database you want to open. You can then use this file handle to refer to your database connection.  Now that we've connected to MySQL, and connected to a database, it's time to close the connection.   Step 3 - Close the connection Closing a connection to a database is quite easy. If you've used a file handle, as above, you just do this:  **mysql\_close($db\_handle);**  Otherwise, you don't need to bother. It's recommended that you take the file handle approach, though. That's what we'll be doing from now on.  So, we'll add a line to close our connection. Here what your code should now look like:  **<?PHP**  **$user\_name = "root"; $password = ""; $database = "addressbook"; $server = "127.0.0.1";**  **$db\_handle = mysql\_connect($server, $user\_name, $password);**  **$db\_found = mysql\_select\_db($database, $db\_handle);**  **if ($db\_found) { print "Database Found "; mysql\_close($db\_handle); } else { print "Database NOT Found "; }**  **?>**  Now that we've got a connection to the database, it's time to look at how you can access the data in the database. We'll do that in the next part.  **PHP Tutorials** Reading Records from a MySQL Database with PHP To read records from a database, the technique is usually to loop round and find the ones you want. To specify which records you want, you use something called SQL. This stands for Structured Query Language. This is a natural, non-coding language that uses words like SELECT and WHERE. At it's simplest level, it's fairly straightforward. But the more complex the database, the more trickier the SQL is. We'll start with something simple though.  What we want to do, now that we have a connection to our database, is to read all the records, and print them out to the page. Here's some new code, added to [the PHP script you already have](http://www.homeandlearn.co.uk/php/php13p1.html). The new lines are in red:  **<?PHP**  **$user\_name = "root"; $password = ""; $database = "addressbook"; $server = "127.0.0.1";**  **$db\_handle = mysql\_connect($server, $user\_name, $password); $db\_found = mysql\_select\_db($database, $db\_handle);**  **if ($db\_found) {**  **$SQL = "SELECT \* FROM tb\_address\_book"; $result = mysql\_query($SQL);**  **while ($db\_field = mysql\_fetch\_assoc($result)) { print $db\_field['ID'] . "<BR>"; print $db\_field['First\_Name'] . "<BR>"; print $db\_field['Surname'] . "<BR>"; print $db\_field['Address'] . "<BR>"; }**  **mysql\_close($db\_handle);**  **} else { print "Database NOT Found "; mysql\_close($db\_handle); }**  **?>**  Before we go through the new code to see what's happening, run your script. You should find that the address you added in [a previous section](http://www.homeandlearn.co.uk/php/php12p4.html) is printed out. (We only have one record at the moment.)  **1 Test Name 12 Test Street**  The first line in the new code is this:  **$SQL = "SELECT \* FROM tb\_address\_book";**  The **$SQL** is just a normal variable. But we're putting into it a long string. This is a **SQL statement**. Here's a brief run down on SQL.   Structured Query Language SQL (pronounced SEEKwel), is a way to query and manipulate databases. The basics are quite easy to learn. If you want to grab all of the records from a table in a database, you use the **SELECT** word. Like this:  **SELECT** \* **FROM** Table\_Name  SQL is not case sensitive, so the above line could be written:  **Select** \* **From** Table\_Name  But your SQL statements are easier to read if you type the keywords in uppercase letters. The keywords in the lines above are **SELECT** and **FROM**. The asterisk (\*) means "All Records". **Table\_Name** is the name of a table in your database. So the whole line reads:  "**SELECT** all the records **FROM** the table called Table\_Name"  You don’t have to select all the records from your database. You can just select the columns that you need. For example, if we wanted to select just the first name and surname columns from this table, we can specify that in our SQL String:  "**SELECT** First\_Name, Surname **FROM** tb\_address\_book";  When this SQL statement is executed, only the **First\_Name** and **Surname** columns from the database will be returned.  There are a lot more SQL commands to get used to, and you'll meet more of them as you go along. For now, we're just selecting all the records from our table.   Back to the Code The first line of our code, then, was this:  **$SQL = "SELECT \* FROM tb\_address\_book";**  SO we have a SQL statement, but we need to pass it to another inbuilt function:  **mysql\_query( )**  The **mysql\_query**( ) function is used to send a SQL query to your database. If you have typed out your SQL correctly, then the function will return a value. This value will be true, false, or a file handle. Because we're using the **SELECT** keyword, the value returned by will be a file handle. In our code, the line was this:  **$result = mysql\_query($SQL);**  The file handle returned in our **$result** variable just points to the results. It doesn't actually bring anything back. To bring back the data, we had this inside a while loop:  **$db\_field = mysql\_fetch\_assoc($result);**  The inbuilt function we're using to bring results back is this:  **mysql\_fetch\_assoc($result)**  The **assoc** part means Associative. As in "associative array". So we're asking that the results be brought back in an array format. In between the round brackets of **mysql\_fetch\_assoc** we have typed the name of our file handle – the one that was pointing to the results of SQL statement.  Remember: an associative array is one where the keys are text. So it's this format:  **Array['One'] =  Array['Two'] = Array['Three]' =**  And not this:  **Array[1] =  Array[2] = Array[3] =**  When the **mysql\_fetch\_assoc** function returns an array, we're putting it all into a variable called **$db\_field**. The Key part of the array is all the Column names from our database tables. This is done automatically for you. So the array format will be this:  **$db\_field[Column\_Name] = Value**  The reason why you're doing this is so that you can loop round the array and access the values from the table. Here's our loop, without anything between the round brackets:  **while ( ) {**  **print $db\_field['ID'] . "<BR>"; print $db\_field['First\_Name'] . "<BR>"; print $db\_field['Surname'] . "<BR>"; print $db\_field['Address'] . "<BR>";**  **}**  So we're printing whatever the value is in the array position **$db\_field['**ID**'],$db\_field['**First\_Name**']**, **$db\_field['**Surname**']** and **$db\_field['**Address**']**. We're also adding a HTML line break at the end, just for printing purposes.  If all that is confusing, just remember the format:  **Array\_Name[Table\_Coulmn\_Name] = Value\_From\_Record**  Our whole while loop, then, is this:  **while ($db\_field = mysql\_fetch\_assoc($result) ) {**  **print $db\_field['ID'] . "<BR>"; print $db\_field['First\_Name'] . "<BR>"; print $db\_field['Surname'] . "<BR>"; print $db\_field['Address'] . "<BR>";**  **}**  Because that is a bit complex, let's go through the steps we've used to access the records from our table:   1. Set up a **SQL Statement** that can be used to get the records from the database table 2. Use **mysql\_query( )**to bring back the records we've specified in Step 1 3. Use **mysql\_fetch\_assoc( )** to set up an array. The array will contain all the records that were returned in Step 2 4. Loop round all the data in the array using a While loop   Step 1 was this, in the code:  **$SQL = "SELECT \* FROM tb\_address\_book";**  Step 2 was this:  **$result = mysql\_query($SQL);**  Step 3 was this:  **$db\_field = mysql\_fetch\_assoc($result)**  And Step 4 was this:  **while ($db\_field = mysql\_fetch\_assoc($result) ) {**  **print $db\_field['ID'] . "<BR>"; print $db\_field['First\_Name'] . "<BR>"; print $db\_field['Surname'] . "<BR>"; print $db\_field['Address'] . "<BR>";**  **}**    If you're still confused, study the code and go over this section. In the next section, we'll adapt the code to add more records to our database table.  **PHP Tutorials** Adding Records to a MySQL Database with PHP To add records to a table in your database, you use more or less the same code as [previously](http://www.homeandlearn.co.uk/php/php13p2.html). The only thing that needs to change is your SQL statement. The steps we're going to be taking are these:   1. Open a connection to MySQL 2. Specify the database we want to open 3. Set up a SQL Statement that can be used to add records to the database table 4. Use mysql\_query( ) again, but this time to add records to the table 5. Close the connection   We've already done steps 1 and 2 on the list. So we can move straight to Steps 3 to 5  Set up a SQL Statement to add records to the database In [our previous script,](http://www.homeandlearn.co.uk/php/php13p2.html) we used some SQL to grab records from our Address Book database table. We then used a While loop to print all the records out. Because we're now going to be adding records to the Address Book table, we need some different SQL. Here's the script. The new line is in blue (The double and single quotes need to be entered exactly, otherwise you'll get errors when you run the code):  **<?PHP**  **$user\_name = "root"; $password = ""; $database = "addressbook"; $server = "127.0.0.1";**  **$db\_handle = mysql\_connect($server, $user\_name, $password); $db\_found = mysql\_select\_db($database, $db\_handle);**  **if ($db\_found) {**  **$SQL = "INSERT INTO tb\_address\_book (First\_Name, Surname, Address) VALUES ('bill', 'gates', 'Microsoft')";**  **$result = mysql\_query($SQL);**  **mysql\_close($db\_handle);**  **print "Records added to the database"; } else { print "Database NOT Found "; mysql\_close($db\_handle); }**  **?>**  You met all of this code from the previous section. The only difference is the new SQL statement! What the code does is to set up some variables, open a connection to the database, and then execute the SQL query. Let's have a look at the new, and rather long, statement.  INSERT INTO … VALUES To add records to your database, you can use the **INSERT** statement. There are plenty of ways to use this statement, but we'll stick with something simple: adding  new values to all of our table columns.  You start by typing the words "**INSERT INTO**". This can be in any case you like: upper, lower or a mix. It's easier for you to read if it's in uppercase letters.  The next thing you need is the name of a table to insert your new values into. For us, this is the table that we've called **tb\_address\_book**.  Following the name of your table, type a pair of round brackets. Inside the round brackets, you can type the names of the columns in your table:  **INSERT INTO** tb\_address\_book (**First\_Name, Surname, Address**)  Notice how we haven't included the ID column from our table. That's because the ID column was the one we set up to be an auto-incrementing number. We don't need to worry about this column because MySQL will take care of adding 1 to this field for us.  Now that you've specified which table you want to insert values into, and specified your column names, you can add the values you want to insert.  To add values, you type the word "**VALUES**" after the round brackets of your column names:  INSERT INTO tb\_address\_book (First\_Name, Surname, Address)**VALUES**  After the word "**VALUES**", you type another pair of round brackets. Inside of these brackets, you can type your values. Each value should be separated by a comma. You can use either direct text, like we've done, or variables. You can even get these values straight from your HTML form, which we'll see how to do later.  So our whole line reads:  **$SQL = "INSERT INTO tb\_address\_book (First\_Name, Surname, Address) VALUES ('bill', 'gates', 'Microsoft')";**  Notice how we've surrounded all of our text with double quotes. But inside of the values round brackets, we've used single quotes.  The syntax is really this (The SQL keywords are in blue):  **INSERT INTO table\_name ( Columns ) VALUES ( values for columns)**  But try your code out now, and see if it's all working properly. You should find that you now have two records in your database table.  **Exercise** Replace the values 'bill', 'gates', and 'Microsoft' with values of your own. Run your script again to add your new record to the database. Now run your other script to read the values back out.    In the next part, you'll learn about something called Magic Quotes.  **PHP Tutorials** PHP and Magic Quotes You can use a HTML form to query your databases. But there are special security considerations you need to bear in mind. We'll look at those issues in this section.  If you use things like text boxes and text areas on your forms, you need to take care. This is because of an attacks like SQL injection. Things like single quotes need to be escaped. But you can use an inbuilt PHP function for this:  **mysql\_real\_escape\_string( )**  We'll see how this works in a moment, but let's get some practical work done. There is a file amongst the [ones you downloaded](http://www.homeandlearn.co.uk/downloads.html) called **magicTest.php** (in the scripts folder). Load this script in your browser, with your server running. You should see a text box and a button. Typed the following name into the text box:  **O'Connor**  Now click the button. You should see the name printed exactly as it is in the text box.  So far, so good. Now, try this.  When you installed your server, there will be a file called **php.ini**. This is a list of all the various settings to do with PHP itself. Locate this file called php.ini (in the folder called **apache**, or do a search for it). Open it up in a text editor. Search for this line:  **magic\_quotes\_gpc = Off**  Change the Off to On, if it's not already on. Then save the changes.  Now load up the your PHP script with the text box and the button. With O'Connor still in the text box, click your button again. You should see this printed:  **O \' Connor**  So PHP has put a backslash before the single quote. But what's going on?  Characters like single and double quotes can be very dangerous, if you're running SQL on your databases tables. These characters can be used to launch a SQL injection attack. So the makers of PHP came up with a function called **magic\_quotes\_gpc**. If this is set to On, then PHP will add the backslash to all single and double quotes. That way, an attacker's life is made more difficult. As an example, we'll load up a database and a script. These are already prepared for you.  Amongst the [files you downloaded](http://www.homeandlearn.co.uk/downloads.html) there is a folder called **databases**. Inside this folder there is a one called **membertest**. Save the entire membertest folder to your **data** directory in your mysql data folder. For Wampserver users this will be at:  **C:\wamp\bin\mysql\mysql5.5.8\data**  (If you have an earlier or later version, the number will be different.)  Now set **magic\_quotes\_gpc = On** back to **magic\_quotes\_gpc = Off**in your **php.ini**file.  Along with the database folder there is a PHP script called **magicTest2.php** (in the scripts folder). We'll use this script, and the database, to teach you about SQL injection. Not so that you can launch your own attacks, of course! It's so that you can thwart them.  **PHP Tutorials** SQL injection and PHP (This lessons follows on from [the previous one](http://www.homeandlearn.co.uk/php/php13p4.html).)  When you open the **magicTest2.php** page in your browser, you'll see three textboxes: one for a username, one for a password, and one for an email address. There is also a button on the form.  Enter the following in the email address text box:  **test1@test1.com**  Click the button, and you should see following print out:  **1 test1 test1 test1@test1.com**  These correspond to the four fields in the database. The four fields are:  **ID username password email**  So the username is **test1**, the password is **test1**, and the email address is**test1@test1.com**.  Now, suppose you were naïve enough to have a database table exactly like that one. An attacker will test to see if any syntax error messages can be returned. If so, this means that the author of the script has not dealt with single/double quotes correctly. The attacker can then go ahead with further probes.  Try your script again. Only this time, add a single quote to the end of the test email address in the textbox:  **test1@test1.com'**  Now click the Submit button. What you should find is that an error message is indeed returned. Something like this:  **Warning: mysql\_fetch\_assoc(): supplied argument is not a valid MySQL result resource**  Because Magic Quotes are off, that single quote is not being escaped. The line in our new script that is doing the damage is the one:  **$SQL = "SELECT \* FROM members WHERE email = '$email' ";**  The SQL this time has a **WHERE** clause added. The WHERE clause is used when you want to limit the results to only records that you need. After the word "WHERE", you type a column name from your database (**email**, in our case). You then have an equals sign, followed by the value you want to check. The value we want to check is coming from the variable called **$email**. This is surrounded with single quotes.  When an email address is entered in the text box on our form, this value goes straight into the variable without any checks. When you type that extra single quote on the end, that will be added to the SQL. This is then run on the database. Because it's a stray single quote, you'll get a syntax error. It's this syntax error that an attacker is looking for.  Next, the attacker will try to add some SQL to yours. Try this. In the email address textbox, type the following. Type it exactly as it is, with the single quotes:  **hi' OR 'x'='x**  When you click the Submit button, you should find that there are no errors, and that the username, password and email address are printed out!  The attacker is trying to find out whether or not the SQL can be manipulated. If the answer is yes, further attacks will be launched. Can the table and field names be guessed? Can a username and password be guessed? It's this kind of attack that you want to thwart.  Try this last one. Enter the following into the email address box:  **' OR ''='**  Now click Submit.  Again, the details are printed out. This is because an **OR** clause has been added. The OR clause is set to a blank string. Meaning that the records will be brought back if it's a valid email address or not!  To stop this kind of attack, you MUST use some inbuilt PHP functions. The one to use for this kind of attack is:  **mysql\_real\_escape\_string( )**  Between the round brackets, you type the string you need to check, followed by an optional database handle. To test this out, there is another script like the one you've just tried. This one is called **magicTest3.php** (in the same scripts folder). If you open this up in your text editor, you should see this added to the code:  **$email = mysql\_real\_escape\_string($email, $db\_handle);**  Now, the $email variable is being checked for any of the following:  \x00 \n \r \ ' " \x1a  If any of the above characters are found, a backslash is added. Try the new script. Enter the following in the email address text box (with the single quote on the end):  **test1@test1.com'**  What you should find is that the following gets returned:  **test1@test1.com\'**  So the single quote has had a backslash added to it. The point is that the dangerous SQL doesn't get executed. Try the above attacks again. This time, you shouldn't be able to get in, if any of the listed escape characters have been used.  But you need to use the function on all variables or data that will be used in your SQL. So you should do this kind of thing:  **$username = mysql\_real\_escape\_string($username, $db\_handle); $password = mysql\_real\_escape\_string($password, $db\_handle); $email = mysql\_real\_escape\_string($email, $db\_handle);**  Examine the code in the new script. Pay attention to where the new lines go: after you have opened a connection to your database.  The PHP manual recommends the following sample script, when working with SQL (all comments are theirs; bold and colour is ours):  [The PHP manual script](http://www.homeandlearn.co.uk/php/quoteSmart.htm)  We have adapted the **magicTest3** script, with the recommended code added, so that you can see it in action. The new script is [magicTest4.php](http://www.homeandlearn.co.uk/downloads.html). Open the script and study the code. See if you can figure out how the new additions work.  As well as using mysql\_real\_escape\_string( ), you'll need to use the other function you saw earlier, in the forms section - **htmlspecialchars**().  It can be a lot of work, dealing with SQL injection attacks, and handling all those escape characters. But if you don't want your databases attacked, you HAVE to defend yourself!   Limit the charcters that a user can enter Another security technique that some advocate is to limit the characters that can be entered. For example, you might have this in your script:  **$valid\_chars = "abcdefghijklmnopqrstuvwxyz"; $valid\_nums = "1234567890"; $valid\_other = "£$^&\_@#~";**  You can then use some Conditional Logic to test if the character the user entered was on your list. If it's not, then you can display an error message.  An excellent walkthrough of security blunders can be found at:  <http://www.sitepoint.com/article/php-security-blunders>  In the next part, we'll take a look at how to create a table using SQL.  **PHP Tutorials** Create a Database Table using SQL and PHP You can create tables using SQL (and whole databases), and specify the fields you want to go in the table. However, doing it this way is not recommended: you tend to forget which fields are in the table, their data types, which field is the primary keys, and which ones are set to NULL values. If you can get to grips with visual tools like phpMyAdmin then so much the better.  To create a table then, you use the CREATE keyword (known as a clause, in database speak). Here's the SQL to create the simple address book we've been using. This assumes that the database itself already exists, and that the PHP code to open a connection has already been written (you saw how to do this in a previous section):  **$SQL="CREATE TABLE AddressBook  ( ID int(7) NOT NULL auto\_increment, First\_Name varchar(50) NOT NULL, Surname varchar(50) NOT NULL, email varchar(50), PRIMARY KEY (ID), UNIQUE id (ID) )";**  **mysql\_query($SQL);**  So you start with the Clause **CREATE TABLE**. Then you type the name of the table you want to create. In between round brackets, you type the name of your table Columns, followed by some formatting. In the code above, the first field being set up is this:  **ID int(7) NOT NULL auto\_increment,**  The column name will be ID. The data type is an integer that is no longer that 7 digits. NOT NULL means you want something in this field, and that it can't be left blank. The ID number will be auto incremented, whenever a new record is added. Notice that there is only one comma in the line. The comma separates each field you want to create in your table.  We're also setting up three other columns here: First\_Name, Surname, and email. First\_Name and Surname can't be left blank ( NOT NULL), but email can be.  At the end, we have these two lines:  **PRIMARY KEY (ID), UNIQUE id (ID)**  The primary key is used for things like joining data from one table to the data from another. We've set this to our ID field. Primary keys don't have duplicate values, so we've set this to be a UNIQUE field.  Once you've written your SQL statement, you can go ahead and execute it:  **mysql\_query($SQL);**  Creating tables like this means a lot of extra, tricky work for you as a PHP programmer. If you can use a tool to do the job for you, then your coding life gets easier!  In the next part, you'll see how to the Update command works.  **PHP Tutorials** Updating a Record in a Database Table using SQL and PHP You can also update a record in your table. Not surprisingly, the word UPDATE is used for this. Here's an example:  **$SQL = "UPDATE AddressBook SET email = 'new\_email\_address' WHERE First\_Name = 'Bill' AND Surname = 'Gates'";**  After the word **UPDATE**, you need the name of the table you want to update. Then you need another Keyword: **SET**. After the word SET, you type the name of the Column you want to change. In the SQL above, we're changing the email column. But notice the**WHERE** clause. We've specified that the record to change should have the First\_Name of Bill and the Surname of Gates.  You can also update an entire column, and change all the values:  **UPDATE AddressBook SET Surname = LOWER(Surname);**  Again, we've specified that the AddressBook table should be updated. We've **SET** the column name as Surname. After an equals sign, we've used the inbuild SQL function LOWER( ). This changes a value to lower case letters. In between the round brackets of the function, we've typed the column name again. This will ensure that all the text in the Surname column gets changed to lower case.  Iin the next part, learn how to delete a record from a database table.  **PHP Tutorials** Deleting a Record in a Database Table with SQL and PHP If you want to delete a record in a table, use the **DELETE** Keyword. Like this:  **$SQL = "DELETE FROM AddressBook WHERE First\_Name = 'Bill' AND Surname = 'Gates'";**  After the **DELETE** word, you need FROM. Then you type the name of the table. Next, you need to specify which record you want to delete. It's a good idea to make sure your **WHERE** clause is going to be a unique value. In the code above, we might have more than one Bill Gates in the table. If we do, everybody called Bill Gates will be deleted! A better solution is to use a unique field from your table, such as an ID field:  **$SQL = "DELETE FROM AddressBook WHERE ID = '7' ";**  Now, only the record that has number 7 in the ID field will be deleted.    In the next part, we'll take a closer look at WHERE.  **PHP Tutorials** Using WHERE to limit the data returned You can add a **WHERE** part to your SQL. But before you do, make sure you read the security section.  Using WHERE limits the records returned from a SQL statement. Most of the time, you don't want to return all the records from your table. Especially if you have a large number of records. This will just slow things down unnecessarily. Instead, use WHERE. In the SQL below, we're using WHERE to bring back only the matching records from the AddressBook table.  **$SQL = "SELECT \* FROM AddressBook WHERE email = 'me@me.com' ";**  When the following code is run, only the records that have an email field of me@me.com will be returned.  You can specify more fields in your WHERE clause:  **$SQL = "SELECT \* FROM AddressBook WHERE First\_Name = 'Bill'AND Surname = 'Gates'";**  In the SQL statement above, we've used the AND operator as well. Only records that have First\_Name value of Bill AND a Surname value of Gates will be returned.  You can also use the operators you saw in the variables section:  **$SQL = "SELECT \* FROM AddressBook WHERE ID >= '10' ";**  In this SQL statement, we're specifying that all the records from the AddressBook table should be returned **WHERE** the ID column is greater than or equal to 10.  Getting the hang of WHERE can really speed up your database access, and is well worth the effort. An awareness of the security issues involved is also a must.  In the next sections, we'll take you through some fuller projects, and explain the code, and the things you need to consider when working on bigger projects like this. First up is a username and password system.  **PHP Tutorials** Adding a PHP Username and Password section - Introduction A lot of sites add a members section, where users are authenticated by means of a username and password. Once the user is logged in successfully, he or she can then gain access to the restricted areas of the site. You'll see how to do that in this walkthrough.  For the walkthrough, there are some scripts already prepared. These can be found in the login folder, which is one of the [folders you downloaded](http://www.homeandlearn.co.uk/downloads.html). (It's in the scripts folder.)  In this walkthrough, we'll explore some of the things you need to bear in mind when creating a username/password section on your site. What we don't offer is a complete login script. This is, after all, a tutorial site, and we'd much rather help you develop your own scripts. Other things you need to consider before implementing a login section on your site are discussed at the end of this section. You need to read these!  A word about the database used for this section The database we've set up for these tutorials is as simple as they come. It consists of a table called **login**, with three fields: an **ID** field, a field for the **username** (called L1), and a field for the **password** (called L2). We're using a simple table, so as not to complicate the tutorials. But you'll want to add more fields, when you come to implement your own login pages. The database can be found in the databases folder. Copy this to the data directory of your mysql folder, just like you did before.  The database table has a test username and password set, so that you can try it out. The username is **usernameTest** and the password is **passwordTest**. Also, make sure cookies are enabled in your browser. You'll see why, later.  But click the next part to get started.  **PHP Tutorials** Add a PHP login page to your site *This lesson is part of an ongoing User Authentication tutorial. The first part is here:*[*User Authentication*](http://www.homeandlearn.co.uk/php/php14p1.html)*along with all the files you need.*    The first script to take a look at in your login folder is **login.php**. Open up this script in a text editor, and well see how it works. Of course, you can fire up your server and try it out. What you'll see is a simple login page with textboxes for username and password, as well as a submit button. It will look like this:  The login page  The HTML for this form can be seen at the bottom of the **login.php** script that you have (hopefully) by now opened. There's nothing special about it. But notice that there's a PHP print statement in the HTML Body section:  **<?PHP print $errorMessage;?>**  This is for displaying error messages for the user.  The first few line of the script, though, just set up some variables:  **$uname = ""; $pword = ""; $errorMessage = ""; $num\_rows = 0;**  The **$errorMessage** variable is an important one. We'll add something to this variable, if an error occurs. We'll then check to see if it's blank or not.  The next part of the code is just the SQL checking function you met earlier. This aims to prevent SQL injection attacks. After this code, we check to see if the form has been POSTED or not (was the Submit button clicked):  **if ($\_SERVER['REQUEST\_METHOD'] = = 'POST') {**  **}**  Most of our code goes between the curly brackets of this if statement. The first thing to do is to get the username and password from the textboxes:  **$uname = $\_POST['username']; $pword = $\_POST['password'];**  We then deal with any unwanted HTML (scripting attacks):  **$uname = htmlspecialchars($uname); $pword = htmlspecialchars($pword);**  Next, we attempt to connect to the database:  **$user\_name = "root"; $pass\_word = ""; $database = "login"; $server = "127.0.0.1";**  **$db\_handle = mysql\_connect($server, $user\_name, $pass\_word); $db\_found = mysql\_select\_db($database, $db\_handle);**  If the database was found, then the variable called $db\_found will be true. We check for this in the next lines:  **if ($db\_found) {**  **} else { $errorMessage = "Error logging on"; }**  If the database isn't found, then some text is added to the error message variable. If the database was found, strip the incoming text of any unwanted characters (SQL Injection attacks). These next two lines call the function at the top of the code:  **$uname = quote\_smart($uname, $db\_handle); $pword = quote\_smart($pword, $db\_handle);**  With the username and password sanitised, we can then set up a SQL command. We're selecting all the records in the database where the incoming username and password match the database table fields called L1 and L2:  **$SQL = "SELECT \* FROM login WHERE L1 = $uname AND L2 = $pword";**  Next, issue the SQL command using mysql\_query( ):  **$result = mysql\_query($SQL);**  We need to check what is returned by the mysql\_query() function. The value in **$result**will either be true (if any records are returned) or false (if none are returned). We're checking to see if there were any errors when the SQL command was issued against the database table. If so, put something in the error message variable:  **if ($result) {**  **} else { $errorMessage = "Error logging on"; }**  If the SQL command was issued successfully, you can see how many rows were returned from the database table. The inbuilt function mysql\_num\_rows( ) is used for this. If no rows were returned, then that tells you that there's something wrong with either the username or password.  **$num\_rows = mysql\_num\_rows($result);**  Next, we test the $num\_rows variable to see if it's greater than zero. If it is, then you have a successful logon. If not, then it's invalid.  **if ($num\_rows > 0) { $errorMessage= "logged on "; } else { $errorMessage= "Invalid Logon"; }**  In the above code, the number of rows returned could be greater than 1. That would mean that 2 or more people have the same username and password. If you have a website where each user has to be unique, then you obviously want to check if $num\_rows = 1. For some websites, it doesn't really matter if 2 or more people have the same login details. But for things like forums, where people are posting and replying to the input of others, then it does matter. After all, you want to credit forum users with the correct posts. For the purpose of this tutorial, assume that it doesn't matter if login details are the same.   Setting a Session So that a user can be remembered across different web pages, you can use something called a Session. A session is simply the time spent at a particular site or sites. You can store values with sessions, and these values will be available to all pages on the site. When you close your browser, the sessions will end. There are quite a lot of ways to use sessions, but we're only interested in saving a value so that it can be referred to across different pages.  In the previous code, the part that checked if the user was OK was this:  **if ($num\_rows > 0) { $errorMessage= "logged on "; } else { $errorMessage= "Invalid Logon"; }**  The code checks to see if the number of rows returned from a SQL command is greater than zero. If it is, then the user is OK. But the code in the login.php script is slightly different. It's this:  **if ($num\_rows > 0) { session\_start(); $\_SESSION['login'] = "1"; header ("Location: page1.php"); }**  What the code does is to set up a session variable. The value in the variable will be 1, if the user logs on successfully. To set up a session variable, you need to issue the start command:  **session\_start( );**  This starts a PHP session. To set up a session variable that you can use to store values, you use this:  **$\_SESSION[ ]**  In between the square brackets of $\_SESSION, you type the name of your variable. Like all variable names, you can call it almost anything you like. Storing values in the session variable is just the same as storing values in a normal variable:  **$\_SESSION['login'] = "1";**  After the script runs, you'll have a session variable called 'login' that is set to a value of 1, if the user is OK. You can then use the "header" function to redirect the user to the page on your site for members, page1.php in the code above.  **header ("Location: page1.php");**  For the else part of the if statement, the code is this:  **if ($num\_rows > 0) { session\_start(); $\_SESSION['login'] = "1"; header ("Location: page1.php"); } else { $errorMessage = "Invalid Login"; session\_start(); $\_SESSION['login'] = ''; }**  Here, we add something to the error message variable:  **$errorMessage = "Invalid Login";**  Next, we issue the "start session" command:  **session\_start();**  But the next line puts something different in to the session variable:  **$\_SESSION['login'] = '';**  We're using the same session name (login), but this time we set it to a blank string. If the user tries to gain access to a restricted part of the site, we'll check for a blank string. A blank string means that the user hasn't logged on successfully, so we'll redirect them to the login page.  A note of caution here. If you switch cookies off in your browser, the script above refuses to work! This is because when you use session\_start, PHP sends the browser something called a session ID. This is a long string of letters and numbers. PHP attempts to save the session ID as a cookie. But it only does this if a line in php.ini from your Apache server is set. This line:  **session.use\_cookies = 1**  If you set this value to 0, then you should be able to log on whether cookies are set or not. The problem is, there's a good chance that you can't set this to zero. Especially if you have web hosting with someone else. The solution, in that case, is to check whether cookies are enabled or not.  You don't actually need PHP to check if cookies are enabled. You can use Javascript. Try this search string in Google:  javascript +cookies +enabled  You should then find plenty of ideas for scripts. Of course, you'd first need to check if Javascript is enabled!  And that's about it for the login script. Here's a run down on what we did:   * Got the username and password from textboxes on a form * Opened a connection to a database * Validated the username and password * Checked to see if any rows were returned from the database * If rows were returned, set a session variable to 1 * If no rows were returned, set a session variable to a blank string * Built up an error message throughout the code   But the point about setting a session variable is so that you can then check its value when users go to other pages on your site. We'll see how to do that now.  **PHP Tutorials** PHP Code to Check if the user is logged on *This lesson is part of an ongoing User Authentication tutorial. The first part is here:*[*User Authentication*](http://www.homeandlearn.co.uk/php/php14p1.html)*along with all the files you need.*    On all pages of your site that you want to secure, you'll need to check if the user was successfully logged on or not. After all, what's to stop non members from simply typing the address of the page in their browsers? If you haven't set any checks, then the page will load, whether they are a member or not. To stop this happening, you can check the session variable that you set up on the login page.  If you open up the page called **page1.php** (in your **scripts** folder), you'll see this complex code at the top:  **<?PHP**  **session\_start();**  **if (!(isset($\_SESSION['login']) && $\_SESSION['login'] != '')) { header ("Location: login.php"); }**  **?>**  This checks to see if the session called login is set, and that it's not a blank string. If it is, then the user is redirected to the login page. In the script, you first start the session:  **session\_start();**  Next comes a complex If statement:  **if ( ) { header ("Location: login.php"); }**  In between the round brackets of the If statement, we have the NOT operator. This is followed by the inbuilt **isset**() function:  **if ( !(isset( ) ) {**  **}**  This says, "If NOT isset". Or, "if the value of the isset function is false ... " If the value in the round brackets of isset is indeed false, then the code between the curly brackets { } gets executed. That code, for us, was the redirection line. What we have between the round brackets of isset is this:  **($\_SESSION['login'])**  That's just our session variable from the login page. Is the user has logged in successfully, a value of 1 will be set inside of this variable.  But we also need to check the session variable for a blank string. So we have and AND part to the statement:  **&& $\_SESSION['login'] != ''**  This says, "AND session login DOES NOT EQUAL a blank string". In other words, we check to see if a session variable has been set, and that it's not a blank string.  If everything is OK then the user will see the HTML code below the PHP at the top. If it's not, you can send them somewhere else. But you need to put that PHP code at the top of every page that you want to protect. And it needs to go before any HTML code. You can't put it in the head section, otherwise you'll get "header" errors.    In the next part, you'll how to let your users log out.  **PHP Tutorials** PHP Code to Log a User Out *This lesson is part of an ongoing User Authentication tutorial. The first part is here:*[*User Authentication*](http://www.homeandlearn.co.uk/php/php14p1.html)*along with all the files you need.*    You'll also want to ensure that your users can log out successfully. This is quite easy, as the only thing you need to do is to destroy the session. Fortunately, PHP has an inbuilt function for that:  **session\_destroy();**  When this function is executed, all the session variables that you set up for the user will be destroyed. Open up your page called **page2.php**, and take a look at the code. It's just this, though:  **<?PHP**  **session\_start(); session\_destroy();**  **?>**  Quite bizarrely, you have to start the session first, even though you've been using sessions throughout your pages! So that the user can log out, you can add a link on your page1.php page:  **<A HREF = page2.php>Log out</A>**  When the user clicks this link, the log out script will be executed. If you try to reload**page1.php**, you'll be redirected to the login page.  In the next part, we'll take a look at how to register a new member.  **PHP Tutorials** Register a New User with PHP Code *This lesson is part of an ongoing User Authentication tutorial. The first part is here:*[*User Authentication*](http://www.homeandlearn.co.uk/php/php14p1.html)*along with all the files you need.*    If the user is new to the site, and wants to sign up, you can add a link to a "sign up" page. Open up your **signup.php** page, and you'll see some code already there. We'll now walk you through what it all does.  When you open up the code for the **signup.php** page, you'll see quite a lot of it is code that you've already met. It starts with the function that checks for dangerous SQL characters. Then we check that the form has been POSTED. The next lines are these:  **$uname = $\_POST['username']; $pword = $\_POST['password'];**  **$uname = htmlspecialchars($uname); $pword = htmlspecialchars($pword);**  We're just getting the username and password from the form, like we did before, and then checking it for unwanted tags. The next thing you need to do, though, is test that the username and password are of the correct length. You don't want a malicious user trying to inject megabytes of text!  **$uLength = strlen($uname); $pLength = strlen($pword);**  **if ($uLength >= 10 && $uLength <= 20) { $errorMessage = ""; } else { $errorMessage = $errorMessage . "Username must be between 10 and 20 characters" . "<BR>"; }**  **if ($pLength >= 8 && $pLength <= 16) { $errorMessage = ""; } else { $errorMessage = $errorMessage . "Password must be between 8 and 16 characters" . "<BR>"; }**  What we're doing here is using the inbuilt function **strlen**( ) to get the length of the string. We then use if .. else statements to check that the username and password are between certain values. If they are ok, the variable called **$errorMessage** is left blank. If they are not ok, we add some text for the error message.  Before checking the username and password against the database, we can check to see if the error message is blank:  **if ($errorMessage = = "") {**  **}**  If it's blank, then everything is ok. In which case the rest of the code is executed. If it's not OK, then the user will see the text of the error message displayed.  Inside of the if statement for the error message check, we just set up the database code like we did before:  **$user\_name = "root"; $pass\_word = ""; $database = "login"; $server = "127.0.0.1";**  **$db\_handle = mysql\_connect($server, $user\_name, $pass\_word); $db\_found = mysql\_select\_db($database, $db\_handle);**  **if ($db\_found) {**  **}**  We're just checking that the database can be found. If it is, then we need to check if the username has already been taken:  **$SQL = "SELECT \* FROM login WHERE L1 = $uname"; $result = mysql\_query($SQL); $num\_rows = mysql\_num\_rows($result);**  **if ($num\_rows > 0) { $errorMessage = "Username already taken"; }  else { }**  The code attempts to select all the records from the table where a match with the username is found. (L1 is the name of the username field in the table.) If any records are returned, then the variable called **$num\_rows** will be greater than zero. We check the value of **$num\_rows** in an if ... else statement.  If the username has already been taken, then we can add something to the error message variable. (But there are security considerations to bear in mind here. Do you really want to tell a malicious user that a username has already been taken? If it's for a forum, then it's ok: the malicious user can simply read usernames from forum posts. But in that case, perhaps we shouldn't be using a username to log people in?)  If the value in the variable $num\_rows is still zero, then we can go ahead and add the user to the database:  **$SQL = "INSERT INTO login (L1, L2) VALUES ($uname, $pword)";**  **$result = mysql\_query($SQL);**  **mysql\_close($db\_handle);**  Here, we use the SQL command INSERT INTO to add a new record to the database.  After the user has been added to the database, we can then set the session variable:  **session\_start(); $\_SESSION['login'] = "1";**  The session variable called **login** will be set to 1. This means that the user can then start using the site straight away. In fact, we redirect them to a different page on the site:  **header ("Location: page1.php");**  Our new user is now a member!  In the final part of this User Authentication walkthrough, we'll take a look at some other things you'll need to consider.  **PHP Tutorials** Other considerations for your User Authentication pages Our [User Authentication scripts](http://www.homeandlearn.co.uk/php/php14p1.html) are fairly simple, and there other things you'll need to consider. Here's a few of them.  Validation Some sites ask you to provide an email address when signing up. (An alarming number of them!) They then send you your login details and confirmation via email, with perhaps a hyperlink that you need to click on to verify the details. You would then enter the verification code or codes before you can start using the site.  To do this, you would need to add more fields to your database table - an email address field and a verified field. The verified field would be set to NO, by default. You could then check this field from all pages of your site. If it still says NO, then the user hasn't yet confirmed the login details. In which case, don't let them in. The verified field would only get set to YES if the user went to the page mentioned in the email and entered the correct details.  This type of script is more complex to set up, and tends to be more frustrating for the user. And there is always a sneaking suspicion that your email address is being sold off to the nearest spammer!  Passwords If you need to save a password to your database table, then you have to encrypt the details. If you look at the **signup** script, you'll notice the use of this function:  **md5($pword)**  The inbuilt function **md5**() returns a 32-character hexadecimal number, based on the string you type between its round brackets. You then save this "hash" number to your password field. Or do it all in one go, with your SQL statement:  **$SQL = "INSERT INTO login (L1, L2) VALUES ($uname,md5($pword))";**  The L1 field is for the username and the L2 field for the password. The VALUE for the password now goes between the round brackets of **md5**()  When you check the password field on the login page, you'd then do this:  **$SQL = "SELECT \* FROM login WHERE L1 = $uname AND L2 = md5($pword)";**  Again, the password goes between the round brackets of md5(). But storing passwords in encrypted format is highly recommended!  Some more things worth considering on your login/signup pages:  * Test if the users is already logged in. That way, they can't sign up repeatedly without closing down the browser * Set a cookie for logins, instead of using sessions. You then need to write code to read the cookie data back for every protected page on your site. * Collect other information, and store it in your database tables: date and time of login, IP address, etc * User's forget their usernames and password. You'll need a link to send them the details. However, don't forget to add some extra security here! Something like a password reminder (memorable date, favourite teacher, etc) is recommended. * Enumeration attacks are quite a common way for malicious users to try and gain access to your site. This is when the attacker can simply sit at his/her pc screen and enter the username and password over and over again, looking for "error message" clues. To thwart this type of attack, you might want to limit how long a user has to log on to your site. A good way to do this is by setting a session to end after so many minutes. This page is worth exploring, for such script ideas:<http://www.weberdev.com/get_example-4267.html>    Conclusion Although our login/sign up scripts are by no means complete, we hope that they've given you something to think about. In particular that these types of scripts are not as simple as you first thought! There are quite a few ready-made login scripts that will do the job for you, but we hope that you will develop your own!  In the next walkthrough, we'll script a complete survey/poll application.  **PHP Tutorials** Build your own Survey/Poll Application - Introduction The Survey web application consists of 5 PHP files and a database. You'll see what they all do as we go along. But there are separate sections: A section where you can set a question for your visitors; a section that allows visitors to vote on your question; and a section where the results can be viewed.   The Survey itself The survey that your users will see looks like this:  The Survey The Results Page When you click the Vote button, your choice will be recorded in the database. If you click the "View Results" button, you'll see this:  The Survey Results Set a Question for your Survey To set a question for your survey, you simply type the question into textboxes on a form. You then set the options that a visitor can choose from. The Form to set a question looks like this:  [The Form to set a question](http://www.homeandlearn.co.uk/php/php15p1.html) (opens in a new window - 43K)   Getting prepared - Check if your downloaded files work To test this out, locate the folder called **survey**, which is amongst the [files and folders you downloaded](http://www.homeandlearn.co.uk/downloads.html) (in the scripts folder). Copy the entire **survey** folder over to your **www**folder. Now start your server and load up the **setQuestion.php** page in your browser by typing:  **127.0.0.1/survey/setQuestion.php**  Or you can use localhost instead:  **localhost/survey/setQuestion.php**  You should see the web page as above that allows you to set a question. This is just a simple HTML form that has no special formatting applied, and so looks a bit messy! But you can use this to add questions to your Survey database.    But don't click the button on the form yet, or you'll get an error message. First, copy the database folder called surveytest to your mysql/data folder. For Wampserver users this should in the following location;  **C:\wamp\bin\mysql\mysql5.5.8\data**  To see if the database has been copied ok, type this into the address bar of your browser (this assumes that your server is up and running):  **http://localhost/phpMyAdmin/**  What you're trying to do is to bring up the phpMyAdmin page. This one:  http://www.homeandlearn.co.uk/php/images/phpMyAdmin.gif  If you can see the phpMyAdmin page, then click the dropdown list for databases. You should see the surveytest database on the list. We'll look at that in more detail in the next part, because it's the key to understanding how the survey web application works.  **The Database for the Survey**  *This lesson is part of an ongoing Survey/Poll tutorial. The first part is here:*[*Build your own Survey/Poll*](http://www.homeandlearn.co.uk/php/php15p1.html)*, along with all the files you need.*    In the [previous part](http://www.homeandlearn.co.uk/php/php15p1.html) of this lesson, you opened [the phpMyAdmin screen](http://www.homeandlearn.co.uk/php/php15p2.html). With this still open, click on "**Please select a database**". Have a look at the items on the drop down list. You should see one called surveytest:  The surveytest database  If you can't see surveytest there, it means you haven't copied the surveytest folder to the correct place.  If you can see surveytest, select it from the drop down list. You should see the names of two tables appear:  The two tables in the surveytest database  Click on **tblQuestions**, and you'll see the **Structure** for this Table (it's too big to fit on this page, so click below to see it):  [The Structure for the tblQuestions Table](http://www.homeandlearn.co.uk/php/php15p2.html) (opens in a new window - 59K  Under the **Table** heading, you'll see the two tables in this database: **answers** and**tblQuestions**. Click on the **Browse** icon for **tblQuestions**, as in the image below:  Browse the tblQuestions Table  You will be taken to the Field names and Rows in the table:  The questions in the Table  The Field names run from left to right, and are important. They are:  **QID Question qA qB qC**  The **tblQuestions** table above has four rows of data, one for each question. The **QID**field is the one to pay attention to. The values in the sample table are **q1**, **q2**, **q3**, and**q4**. This QID field is the **Primary Key** in this table. This means that the data in this field has to be unique. You can then use this QID field to identify each row in the table. This same field, QID, is also in the **answers** table, along with the **qA**, **qB**, **qC** fields. This allows you to select all the records in both tables based on the QID field. You just pull all the records that match. For example, you can say "Select all the records in both tables where the QID field equals q1".  Take a look at the answers table by clicking the link on the left hand side. Then click on**Browse** at the top. You should see this:  The answers Table  In the answers table, the unique field (the primary key) is the ID field. This is just an auto incrementing number that you used in an earlier section. You don't have to worry about this field. But notice that the **QID** field is also there, along with the same values from the**tblQuestions** table: **q1**, **q2**, **q3**, and **q4**. This matching **QID** field in the **answers** table is something called a foreign key, in database terminology. Joining data from a primary key in one table to a foreign key in another is common technique in database creation. You do this when you want to keep data separate, and to avoid having too many fields in a single table. It also speeds things up. In our example database, we can keep the questions and answers separate.  (NOTE: If you have some knowledge about databases, you'll know about Referential Integrity. Unfortunately, phpMyAdmin doesn't enforce this. So if you delete a row from one table, the corresponding row in another table won't get deleted - you have to code for that yourself!)  The A, B, and C fields in the answers table record how many people voted for each option of your question. So, for question four (q4) 28 people voted for option A, 127 people voted for option B, and 52 people voted for option C. If you look at the matching row (q4) in the tblQuestions table you'll see that the question was: Do you believe in UFOs? (These answers were entered by us - it's not real data!)  Now that you have a good idea about how the database works, let's go through the code that sets a question.  **Setting a question – the code**  *This lesson is part of an ongoing Survey/Poll tutorial. The first part is here:*[*Build your own Survey/Poll*](http://www.homeandlearn.co.uk/php/php15p1.html)*, along with all the files you need.*    In the [previous part](http://www.homeandlearn.co.uk/php/php15p2.html) of this lesson, you had a look at the structure for the Survey database. We're now going to look at the code for setting a question. The web page itself, remember, looks like this:  [**The Form to set a question**](http://www.homeandlearn.co.uk/php/php15p3.html) (opens in a new window - 43K)  Using your favourite text editor, open up the file called **setQuestion.php** again, and take a look at the code. Most of it is code you've already met. In the first two sections we just get the data entered in the textboxes on the form (the questions and three possible options), check for any unwanted script tags, and then open up a connection to the database. The first new code is this:  **//============================================ // GET THE LAST QUESTION NUMBER //============================================**  **$SQL = "Select \* FROM tblQuestions";**  **$result = mysql\_query($SQL); $numRows = mysql\_num\_rows($result);**  **$boolLastRow = mysql\_data\_seek($result, ($numRows - 1));**  **$row = mysql\_fetch\_row($result);**  **$qID = $row[0];**  **$next\_Q\_Number = ltrim($qID, 'q'); $next\_Q\_Number++; $question\_Number = 'q' . $next\_Q\_Number;**  As the comments say, we're getting the last question number from the **tblQuestions**table. We need to do this because the primary key doesn't auto increment.  Because the primary key is a unique field, the auto increment feature of MySQL would have added one to the primary key for us. But we've made our own unique field to be used as a primary key (q1, q2, q3, etc), so we're going to have to increment the values with code. The idea is that we get the last number value from the QID field, then add 1 to it. (For simplicity's sake, there's no checking to see if the value we write back to the database is indeed unique. If it isn't, you'll get an error. You'll see how to code for this in the next section.)  The code, then, starts off by getting all the records from the **tblQuestions** table:  **$SQL = "SELECT \* FROM tblQuestions";**  Next, we try to run this query:  **$result = mysql\_query($SQL);**  We could check here if the value in the variable called **$result** is true. If it wasn't, we could display an error. (Another thing for you to do!) But we're keeping things simple, so you can see how all this works.  The next thing to do is to get the number of rows in the table:  **$numRows = mysql\_num\_rows($result);**  We need to get the number of rows so that we can move the pointer to the last row in the database. We want to get this last row so that we can check the QID field. Remember: we're trying to add 1 to the value in the QID field - to increment it. The code that moves the internal pointer to the last row in the database is this:  **$boolLastRow = mysql\_data\_seek($result, ($numRows - 1));**  We're using an inbuilt PHP function called **mysql\_data\_seek**( ). This allows you to jump to any row in a table. Between the round brackets of the function, you first type the name of your resource (**$result**, for us). Then, after a comma, you put the row that you want to jump to: **$numRows - 1**. The count with **mysql\_data\_seek**( ) starts at zero, so we're deducting one from the number of rows in our table. Otherwise, we'd get an error about no such row found. We're returning all this to a boolean variable called **$boolLastRow**. You can write an error message, if this returns a value of false. But we're leaving out the error checking here.  Once we've moved the pointer to the last row in the table, we can fetch the data back:  **$row = mysql\_fetch\_row($result);**  You've met the inbuilt function **mysql\_fetch\_row**( ) before. It fetches a row of data, and puts it all in to an array. You can then access the array, and manipulate the data.  **$qID = $row[0];**  The QID field is at position zero in the array (it's the first field in the database). So we just use **$row[0]** to return the value. This is then put in to the variable we've called **$qID**. This variable will then hold the last QID value from the table. The format we're using for the QID field is to type a letter "q" followed by a number. To increment the QID field, we can just add 1 to the number then join that number to the letter "q". The next few lines do exactly that:  **$next\_Q\_Number = ltrim($qID, 'q'); $next\_Q\_Number++; $question\_Number = 'q' . $next\_Q\_Number;**  We use the**ltrim** function to strip off the letter "q". This leaves just the number itself. You can then increment this number (**next\_Q\_Number++**). The final line joins the new data back together:  **$question\_Number = 'q' . $next\_Q\_Number;**  This adds the new number to the letter "q", and then stores it in a variable called**$question\_Number**. It's this question number that will get written to the QID fields in both the **tblQuestions** table and the answers table.  The next two sections use the INSERT INTO command to add the new question to the database. The first thing to do is to add the question to the tblQuestions table. The rather long SQL line that does that is this:  **$SQL = "INSERT INTO tblquestions (QID, Question, qA, qB, qC)VALUES ('$question\_Number', '$question', '$answerA', '$answerB', '$answerC')";**  You've met code like this in a previous section. You should be able to figure out what it does: adds the data into the Field names mentioned in the first round brackets.  To update the table, we run the SQL query:  **$result = mysql\_query($SQL);**  The next SQL command is slightly different:  **$SQL = "INSERT INTO answers (QID, A, B, C) VALUES('$question\_Number', 0, 0, 0)";**  Again, it's an INSERT INTO command, but note that we're now updating the answers table. The VALUES between the round brackets are:  **'$question\_Number', 0, 0, 0**  The value inside of **$question\_Number** will be the QID number. But the next three values are all zero. This sets up the answers table, and ensures that the default options are all filled in. A, B and C, remember, will hold the number of votes. We start them off at zero because nobody has voted yet!  And that's about it for setting the question. We've covered quite a lot of ground in a short space of time, so let's review what we did.   1. Created two tables in the same database 2. Had a primary key in one table that is joined to a foreign key in the other table 3. Wrote code to move an internal pointer to the last record in a table 4. Returned a specified row, and incremented a value to be used as a unique key 5. Inserted a new record into two tables, using the same field in both   Probably the most important thing to learn in this section is how to join separate tables together using a primary/foreign key combination. We'll use this technique again when we create a forum. For now, let's move on to the survey itself.  **The Code for the Survey itself**  *This lesson is part of an ongoing Survey/Poll tutorial. The first part is here:*[*Build your own Survey/Poll*](http://www.homeandlearn.co.uk/php/php15p1.html)*, along with all the files you need.*    In the [previous section](http://www.homeandlearn.co.uk/php/php15p3.html), you saw how to code for setting a question in your Survey. We'll now look at the Survey page.  The survey itslef can be found in the survey folder and is called **survey.php**. Open up this page in a browser by typing the following address:  **127.0.0.1/survey/survey.php**  You should see this:  The Survey    All we have here is a question, and three possible answers. To vote, you select an answer and click the button. When you do, you're taken to another page which thanks you for voting. On the page, there is also a button that allows you to view the results. Let's see how it all works.  To see the code for the survey, open up **survey.php** in your text editor. The first line you'll see is this:  **include 'sqlSurvey.php';**  The include file is an important one. Open up this file (also in the survey folder), and you'll see that it's just a SQL command. This SQL is used to pull a question from the database.  The code is this:  **$qNum = 'q4';**  **$SQL = "SELECT \* FROM tblquestions WHERE tblquestions.QID = '$qNum'";**  Only two lines long! The first line sets the question number. This is the QID field from the tblQuestions table. To set a new question, the only thing you have to change is this value. Try it. Change 'q4' to one of the other three values in the table: q3, q2, or q1. Save the file, and reload survey.php in your browser. You should see the question and answers change.  The important part of the SQL line is this:  **WHERE tblquestions.QID = '$qNum'**  We're saying select all the records WHERE the QID field matches the value in the variable called **$qNum**. That's enough to pull the question and answers from the table! But although the file is included on the first line, the code inside of it doesn't get run till a little later.  The next few lines just set up some variables, and put default values in them. After that, we add the code that opens up the database. You've met all this before, so we won't go into it.  The part of the code that uses the include file is this:  **$result = mysql\_query($SQL);**  The variable called**$SQL** is in the include file. The survey code can see this variable because of the include directive. So it knows what's inside of it.  If mysql\_query( ) succeeds, we can put the values into an array with the next line (though you should write could for false values inside of $result):  **$db\_field = mysql\_fetch\_assoc($result);**  The data from the SQL is then placed into an array called **$db\_field**. You can get at these values like this:  **$qID = $db\_field['QID']; $question = $db\_field['Question']; $A = $db\_field['qA']; $B = $db\_field['qB']; $C = $db\_field['qC'];**  Here, we're returning the values from the following fields in the table: QID, Question, qA, qB, and qC. We've put these into variables of their own.  Once we have the question and the answers, we can put these into the HTML on the page. There are two HTML forms on the page. The first is this:  **<FORM NAME ="form1" METHOD ="GET" ACTION ="process.php">**  **<?PHP print $question; ?> <P> <INPUT TYPE = 'Radio' Name ='q' value= 'A' <?PHP print $answerA; ?>><?PHP print $A; ?> <P> <INPUT TYPE = 'Radio' Name ='q' value= 'B' <?PHP print $answerB; ?>><?PHP print $B; ?> <P> <INPUT TYPE = 'Radio' Name ='q' value= 'C' <?PHP print $answerC; ?>><?PHP print $C; ?> <P>**  **<INPUT TYPE = "Submit" Name = "Submit1" VALUE = "Click here to vote">**  **</FORM>**  The question from our code is placed on the page with this line:  **<?PHP print $question; ?>**  This is the value that we got from the field in the tblQuestions table.  The code for the answers is then added to the radio button on the form:  **value= 'A' <?PHP print $answerA; ?>><?PHP print $A; ?>**  The "print $answerA" part will just add a value of checked or unchecked to the radio button, depending on whether it was selected or not. An answer is added to the radio button like this:  **<?PHP print $A; ?>**  Whatever data we pulled from the **A** "field" in the table will end up in the variable called**$A**. This is then printed to the page.  If the button is clicked, though, we're sending it to a page called process.php. We'll take a look at that in a moment, but notice the second Form on the page:  **<FORM NAME ="form2" METHOD ="GET" ACTION ="viewResults.php">**  **<INPUT TYPE = "Submit" Name = "Submit2" VALUE = "View results"> <INPUT TYPE = "Hidden" Name = "h1" VALUE = <?PHP print $qID; ?>>**  **</FORM>**  This is for the "View Results" page. When this button is clicked, it goes to a new page: viewResults.php. But there is a crucial HTML form element in the form code:  **<INPUT TYPE = "Hidden" Name = "h1" VALUE = <?PHP print $qID; ?>>**  This sends the QID number to the viewResults page. We can then use this value to pull the correct records from the two tables. You'll see how this works soon. But one more thing to notice: both forms use the GET method to hand data to the next page. Now let's move on to the code for the **process.php** page.  **Add the Vote to the Database**  *This lesson is part of an ongoing Survey/Poll tutorial. The first part is here:*[*Build your own Survey/Poll*](http://www.homeandlearn.co.uk/php/php15p1.html)*, along with all the files you need.*    When the Vote button is clicked, the user will be sent to a new page. Behind the scenes, you're recording the vote and adding it the database.  As well as adding the vote to the database, you'll probably want to implement some sort of check to prevent people from voting over and over again. If you open up the code for the **process.php** page (in the survey folder), you'll see that the checking is done via a session variable. You met this code during the username/password walkthrough.  **session\_start(); if ((isset($\_SESSION['hasVoted']))) { //Already Voted } else { //Process the Vote }**  (NOTE: Session variables work by sending you an ID as a cookie. If you have cookies disabled then the session variable won't work, and you can vote over and over again! You might want to check that cookies are enabled in the browser.)  So we start a session, and check if the session variable called hasVoted has been set. If it has, then a vote from this user has already been added to the database. In which case, a message is displayed.  If no session has been set, then the else part of the if statement is executed. The first line of this is another if statement:  **if (isset($\_GET['Submit1']) && isset($\_GET['q'])) {**  **}**  Here's we're checking for two things: was the Submit button called Submit1 clicked on the previous page, and has the radio button data been handed over to this page (the radio button data will be in the variable called 'q')? If the answer to both questions is Yes then we can go ahead and process the data; if the answer is No, then we can assume that the button was clicked but the user didn't select a radio button. In which case, this message is displayed: print "You didn't selected a voting option!";  If all went OK, though, the first line of the new if statement to get executed is this:  **$selected\_radio = $\_GET['q'];**  This just gets which radio button was clicked. The value comes from the HTML form, and will be A, B or C. This is then placed into the variable called**$selected\_radio**.  The next few lines open a connection to the database, like we did before. After the database has been successfully opened, we have this:  **$\_SESSION['hasVoted'] = '1';**  This is our "hasVoted" session variable, and we're placing a value of "1" into it. If the user tries to vote again, a message of "You've already voted" will be displayed.  The next line is where we add the record to the answers table in our database:  **$SQL = "UPDATE answers SET $selected\_radio = $selected\_radio + 1";**  Here, we're using the SQL command UPDATE. After the word UPDATE, you type the name of the table you want to UPDATE (answers, for us). We only want to update the field that was passed to the page in the "q" variable. This will be either A, B or C, which are the column names from the answers table. We can then use the SET keyword to set just that column from the table. Because the variable called **$selected\_radio** will contain only A, B or C we can just add 1 to whatever is currently there:  **SET $selected\_radio = $selected\_radio + 1";**  You'll notice that we're not making sure to move to the end of the records in the answers table (which we should do), but trusting MySQL to do it for us.  This is not a good idea! In your own code, you should make sure that the correct record is being updated. We're taking shortcuts for simplicity's sake.  After the table is updated, we close the connection to the database, and print out a message: "Thanks for voting".  And that's it - a vote has been added to the table.  You can add a "back" link to the HTML, or do anything else you like with your page. Or perhaps you could add a link so that the results of the voting can be viewed? We'll take a look at the code for the **results** page now.  **Viewing the Results of the Survey**  *This lesson is part of an ongoing Survey/Poll tutorial. The first part is here:*[*Build your own Survey/Poll*](http://www.homeandlearn.co.uk/php/php15p1.html)*, along with all the files you need.*    On the **survey.php**page, there is a button that can be clicked to View the Results. If this button is clicked, the user will see something like this:  The results of the survey  The red line represents how many people voted for a particular option. After that, we have the number as a percentage, and what the option was.  On the previous page, survey.php, The HTML form was this:  **<FORM NAME ="form2" METHOD ="GET" ACTION ="viewResults.php">**  **<INPUT TYPE = "Submit" Name = "Submit2" VALUE = "View results"> <INPUT TYPE = "Hidden" Name = "h1" VALUE = <?PHP print $qID; ?>>**  **</FORM>**  The value in the variable **$qID**(part of the Hidden HTML form element) is coming from the PHP code you saw earlier. This is the QID field that is in both the questions and answers tables in our database. We need to pass this QID number over to the**viewResults.php** page so that the correct results can be displayed.  If you open up the code for **viewResults.php** page (in the survey folder), you'll see there's quite a lot there! But most of the code is taken up with manipulating the **red.jpg**graphic! We're just stretching a red line, depending on how many votes were cast for an option. The rest of the code pulls the data from the table, and puts values into variables. We'll keep the two separate, for explanation purposes.   How many people voted for a particular option? To see how many people voted for A, B or C, we first have to get that value from the**viewResults.php** page.  **if (isset($\_GET['Submit2'])) {**  **$qNum = $\_GET['h1'];**  **}**  First, we check to see if the Submit button was clicked on the previous page. If it was then we can GET the hidden value:  **$qNum = $\_GET['h1'];**  The Hidden form element had the NAME "**h1**". It's this NAME that goes between the square brackets of $\_GET. The VALUE of the **h1** hidden element is coming from the**$qID** variable. This will be the QID field in the database tables tblQuestions and answers. We can use this value to pull records from the tables, which the next two lines do:  **$SQL = "SELECT \* FROM tblquestions, answers WHEREtblquestions.QID = answers.QID AND answers.QID = '$qNum'";**  **$result = mysql\_query($SQL);**  Here, we're selecting all the records from two tables:  **SELECT \* FROM tblquestions, answers**  But we need a WHERE part. The WHERE clause is quite tricky:  **WHERE tblquestions.QID = answers.QID AND answers.QID = '$qNum'**  To match the two QID fields, we have this:  **tblquestions.QID = answers.QID**  The name of the table goes first, followed by a dot. After the dot, we've typed the QID field. This is present in both tables.  But we only need the records where the QID field matches the qNum that was handed over from the previous page:  **AND answers.QID = '$qNum'**  We're saying "Only bring back the records where the two QID fields match AND where the QID value in the answers table is equal to the value in the variable called $qNum". A bit more complex, but go over it a few times and it will make sense. Hopefully!  After the SQL has been run, we put the results into an array:  **$db\_field = mysql\_fetch\_assoc($result);**  **$question = $db\_field['Question']; $answerA = $db\_field['A']; $answerB = $db\_field['B']; $answerC = $db\_field['C'];**  **$qA = $db\_field['qA']; $qB = $db\_field['qB']; $qC = $db\_field['qC'];**  If you look at the values in the square brackets, you'll see that they are the field names from both tables. We're getting the value for the Question, the answers A, B and C, as well as the original options for each questions (qA, qB and qC). All of these values are then placed into variables, ready for displaying on the page.  We could just print the values to the page, like this:  **print $question; print $answerA; print $answerB; print $answerC;**  But that's not very interesting for the viewer. Better is some form of graphic, like the one at the top of this page. This is what you'll see when you click the button to View the Results. Here's how it works.   Display a graphic for the results At the top of the code, you'll notice some variables are set up:  **$imgTagA = ''; $imgWidthA = '0';**  **$imgTagB = ''; $imgWidthB = '0';**  **$imgTagC = ''; $imgWidthC = '0';**  **$imgHeight = '10'; $totalP = ''; $percentA = '0'; $percentB = '0'; $percentC = '0';**  What we're going to do here is to manipulate the HTML IMG tag. This tag is used to display an image, and ordinarily would look something line this:  **<IMG SRC = 'red.jpg' HEIGHT = '10' WIDTH = '100'>**  The part we want to manipulate with our PHP code is the WIDTH value. Instead of setting it to a fixed value, we can set it to a value from the answers table.  In the answers table, the columns for A, B and C hold how many people voted for a particular option. We have three separate IMG tags in our code, one for each of the columns. If, for example, A held a value of 45 (meaning 45 people voted for this option), we can make the WIDTH of the red line image 45 pixels.  Except, 45 pixels is not very wide. Much better to turn them all in to percentages. Which is what the following code does:  **$totalP = $answerA + $answerB + $answerC;**  **$percentA = (($answerA \* 100) / $totalP); $percentA = floor($percentA);**  **$percentB = (($answerB \* 100) / $totalP); $percentB = floor($percentB);**  **$percentC = (($answerC \* 100) / $totalP); $percentC = floor($percentC);**  **$imgWidthA = $percentA \* 2; $imgWidthB = $percentB \* 2; $imgWidthC = $percentC \* 2;**  First you need to total up how many people voted:  **$totalP = $answerA + $answerB + $answerC;**  Next, you can multiply A, B and C by 100 and then divide by the total:  **$percentA = (($answerA \* 100) / $totalP);**  Because this would give an answer like 24.567, we can use the inbuilt Math function floor to strip the "point something" at the end:  **$percentA = floor($percentA);**  Finally, you can multiply by, say, 2 to give the red line some extra width:  **$imgWidthA = $percentA \* 2;**  The values in $imgWidthA, $imgWidthB and $imgWidthC can then be used to build an IMG tag:  **$imgTagA = "<IMG SRC = 'red.jpg' Height = " . $imgHeight . " WIDTH = " . $imgWidthA . ">";**  To display this on the page, we use the following:  **print $imgTagA . " " . $percentA . "% " . $qA . "<BR>";**  So the red line gets displayed first, followed by the percentage of people who voted for this option. A percent sign and the option itself complete the line.  And that completes the survey walkthrough. Feel free to adapt and amend the code you downloaded. But don't forget to make a backup of the original files!  Next, we'll take a look at how to build your own online Forum.  **Introduction to the Forum**  In this section, you'll see what is involved in building a bigger web application, as we take you through the coding for an online Forum. You may not want a Forum on your site, of course, but the project is well worth doing all the same. There are plenty of techniques here that can be carried over into your own pages, and hopefully you'll acquire some new coding skills. There are plenty of exercises to complete along the way, so it's not all theory!  This Forum, however, is not intended to be an application that you can simply upload to your own site, and get working straight away. It is purely for teaching purposes. All the files for this section can be found in the **forum** folder [you downloaded](http://www.homeandlearn.co.uk/downloads.html). Before you start, it's a good idea to make a copy of this folder. That way, you won't be changing any of the original files. Copy the entire **forum** folder to the www directory on your server. You should then have the forum folder inside of your www folder.  If you double click the **forum** folder, you'll see all the files and scripts mentioned in this walkthrough. The database for this walkthrough can be found in the **databases** folder, and is called **dbforum**. This folder contains the database for this projects, and all the tables. Copy this folder the **data** folder of MySQL.  For Wampserver users this will be in the following location:  **C:\wamp\bin\mysql\mysql5.5.8\data**  To check if everything is working, start your server and navigate to this address:  **127.0.0.1/forum/forumTest.php**  Or you can try this:  **localhost/forum/forumTest.php**  What you should see is the very basic forum in the link below:  [The Basic Forum](http://www.homeandlearn.co.uk/php/php16p1.html) (Opens in a new window 80K).  Obviously, the HTML needs improving! But this is one of things you'll be changing, as we go along. You'll see where you can adapt the HTML, and how to add your own code.  But the basics of the forum in the image above are common to most forums: you have the forum sections as hyperlinks, and then additional information along side each link. In our forum, we'll see how to get the number of posts in a section, and how many people have replied to each post.  Once you have loaded the forum into your browser, play around with the it. Only the Microsoft Word section is working, so click this link. You'll then see this:  [The Posts in the Forum](http://www.homeandlearn.co.uk/php/php16p1.html) (Opens in a new window 110K).  What you're looking at is all the Posts in the Microsoft Word section of the forum. Only members are allowed to Post on the forum, and you'll see the member name in the first column (Posted By). Only one Post has any replies - the first one at the bottom. Click this link, and you'll see a new screen:  [The Replies to the Posts](http://www.homeandlearn.co.uk/php/php16p1.html) (Opens in a new window 125K).  Because there are 11 replies, they are split over 2 pages, with a link to each page. If you were logged in as a member, you'd see a different link at the bottom of the page, allowing you to reply to this Post. Try this for yourself. Click on the link that says "Login Here". You'll be taken to an area that asks you to enter a username and password. You can use any of the username/password combinations that you can find in [your scripts/forum folder](http://www.homeandlearn.co.uk/downloads.html). Open up the text file called **uandp.txt** and you'll see them all in there.  Login with one of the username/password combinations, and you'll be allowed to Post topics in the Microsoft Word section, and Reply to them.  Now that you have a good idea of how the forum works, it time to get started. There's one important thing we need to do first: set up the database.  **Setting up a Database for a bigger project**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need.*    So far, the databases you have been constructing have been fairly simple. They only consisted of one or two tables, and there wasn't much linking done between each table. For this fourm project, we'll need a more complex database. And lots of tables!  To see what the database looks like, enter this address in your browser.  **http://localhost/phpMyAdmin/**  This will open the **phpMyAdmin** page. On the left hand side, click the drop down list for Databases. If you have copied the dbforum database over to the data folder correctly, you should see it on the list:  The phpMyAdmin page  Select **dbforum** from the list and you should see the tables in the database displayed. (If you can't see dbforum, you may not have copied the database over to your[mysql/data folder](http://www.homeandlearn.co.uk/php/php16p1.html) correctly.)  The Tables in the dbforum Databse  In this database, there are a massive 12 tables! But why so many?  The reason why there are so many tables is because we want to access the information in the database as quickly as possible. You can get a real speed boost by keeping data in different tables, rather than lumping all the information into one or two tables. For example, we could have had one table to hold all the information about the Microsoft Word section. We could have had columns for the Posts, and columns for the replies. But this would mean that the one table would be holding a lot of information: the text for the post, and the text for the replies. It's much better (for coding and for speed) to keep posts and replies in separate tables.  (When working on larger web projects, taking the time to plan your database, and all the tables you'll need, will pay dividends. One thing you don't want to do is get half way through your coding only to discover that your database isn't structured correctly, and that you're going to need another table, or extra columns.)  As well as 10 tables for the posts and replies there is a separate table for the forum sections, and a separate table for member information. Let's take a look at the**forumsections** table first:  **The Forum Sections Table**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php16p2.html)*.*    The Forum Sections table is the simplest table in [the database](http://www.homeandlearn.co.uk/php/php16p2.html). It consists of only two fields. In **phpMyAdmin**, click on the **forumsections** table on the left hand side. Then click on Browse, at the top. You should see this:  The fields in the forumsections Table  The two columns (fields) are **sectionID**, and **sections**. If you look at the main page of the forum again, you'll see both the **sectionID** and the **sections** text in the status bar at the bottom of the page:  The Forum page  The **sectionID** is at the end of the link: **sID = secWP**. The **sections** text is the text for the hyperlink itself (the one with the hand over it). But your cursor over the other links and watch what happens in the status bar at the bottom of the page. When the link is clicked, though, we'll be taking the **sID** value over to the next page.    **Exercise** In **phpMyAdmin**, click the **Edit** button (the pencil) next to **secWP**. From the page that appears, change Microsoft Word into anything you like. Click the "Go" button. Now, with the Forum start page displayed in your browser, refresh your page. What happens? You should see the text on the page change, as in the image below:  The link text has changed    So the **sections** text in the **forumsections** table is used for the hyperlink on the main page of the forum. The **sectionID** is used to identify which section of the forum a visitor wants to go to. You'll see how that works soon. But have a look at the members table now.  **The Members Table**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php16p3.html)*.*    The members table, as its name suggests, holds information about people who have joined the forum. Click the **members** table in **phpMyAdmin**, then click **Browse** at the top. You should see the fields in the table:  The fields in the members table  There are five columns in the members table: **memberID**, **username**, **password**,**signupDate**, and **email**. (The passwords, by the way, should be encrypted. For tutorial purposes, however, we haven't done this. But you saw how to do this in an earlier section.)  The unique column (the primary key) is **memberID**. You'll see this column in the other tables, as well. When you're displaying information about a Post or a Reply, you'll want to include which member did the posting or replying. If you match the memberID in, say, the **wpposts** table with the memberID in the **members** table, you then have an easy and efficient way to display the member information. The alternative is to include a username column in the posts and replies table. But this would be a really bad way to construct a database, because you'd be duplicating information. With a single members table, and a reference field in other tables, you don't need to add username fields into every post and reply table. Just link to the members table.  But the members table we have here is really basic. You could add more fields. A common practice in Forums is to reward loyal members who Post and Reply often with a higher status than other members. Its these members who keep a Forum alive. Because they'll give up their spare time, it's a good idea to give them a little control over the Forum, say as a moderator. In the members table, then, you can add a Status field. This would allow you to change what a member is allowed to do.  Depending on what type of forum you want, you can add even more fields to the members section. What about adding an Avatar field? This could be a link to an small image that the member wanted to use in their Posts and Replies. But we're sure that you can think of more fields that can be added to our basic members table!  In the next part, we'll take a look at the 5 Post tables in the database.  **The Post Tables**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php16p4.html)*.*    There are 5 Post tables in the database, one for each section of the forum. Here's what each Post table is called, and what they are for:  **wpposts** - All posts to do with Microsoft Word **vbposts** - All posts to do with Visual Basic .NET **xlposts** - - All posts to do with Microsoft Excel **wdposts** - All posts to do with Web Design **phposts** - All posts to do with PHP  Since all of these tables have the same structure, we'll just take a look at the **wpposts**table. Click on this table in **phpMyAdmin**, then click the **Browse** button. You should see the following:  [**The wpposts table**](http://www.homeandlearn.co.uk/php/php16p5.html) (Opens in a new window 47K).  There are five fields in each of the post tables: threadID, memberID, threadTopic, postText, and datePosted. Here's what they all do:  **threadID** - used to identify an individual post (the primary key) **memberID** - used to identify which member posted this topic **threadTopic** - the text used in the hyperlink when viewing the topics page **postText** - the text for the Post **datePosted** - the date the memebr Posted the topic  We'll access these post tables when a user clicks on a particular forum section on the main page. When they do, this is what the user sees in the browser:  The Post Table  If you hold your mouse over a link, you'll see this in the status bar:  A link for a Post  The thing to notice about the link is **rID=pos1**. The **pos1** is coming from the table, and is the **threadID** field. Hold your mouse over a different link and you'll see the pos1 change. When the link is clicked, we'll be pulling information from the **reply** table. The reply table will also have a **threadID** field. We'll only be displaying records where the threadID fields match.  The final five tables in the database are for the replies. We'll see them next.  **The Reply Tables**   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php16p5.html)*.*    There are 5 Reply tables in our database, one for each section of the forum. Here's the table names, and a short description:  **wpreplies** - All replies for the Microsoft Word section **xlreplies** - All replies for the Microsoft Excel section **phreplies** - All replies for the Microsoft PHP section **vbreplies** - All replies for the Microsoft Visual Basic .NET section **wdreplies** - All replies for the Microsoft Web Design section  Again, all these tables have the same structure. So we'll just examine the **wpreplies**table. In **phpMyAdmin**, click the **wpreplies** table on the left hand side. Then click the**Browse** link at the top of the page. You should see the following:  The wpreplies table  There are five fields in this table: **ID**, **threadID**, **memberID**, **reply**, and **dateReplied**. Here's what they do:  **ID** - the primary Key. An auto incrementing number. **threadID** - used to identify which post a member is replying to **memberID** - used to identify which member replied to a post **reply** - the text of the reply **dateReplied** - the date the meber replied to the post  At the moment, the only Post to have any replies is **pos1**. That's why there are no other values in this field. But if other posts have replies they would be labelled pos2, pos3, etc. The **threadID** of the wpposts table is linked to the threadID of the wpreplies table. You can then say, "Select all the replies where there is a threadID match."  The **memberID** field is also present in this table. This allows us to link to the members table and display the username of the member who replied.  To get a clearer idea of how this database is structured, study the following list: | | | | | | | |  | **Table Name** | **Field Names** | | | | | |  | **forumsections** | sectionID | sections |  |  |  | |  | **members** | memberID | username | password | signupDate | email | |  | **wpposts** | threadID | memberID | threadTopic | postText |  | |  | **wpreplies** | ID | threadID | memberID | reply | dateReplied | | These are the tables we'll be working with, when we do our coding. The **memberID** is in three of the four tables above; the **threadID** is in two of the tables. We'll be using these fields to link information. When you construct your own databases, it's important to consider which fields you can use to link information in different tables. Planning like this can speed things up when you're pulling data from your databases with PHP code.  Now that you have a decent grasp of just how the database is constructed, we can take a look at the code.  **Code Strategies and Objectives**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need.*    There are a lot more PHP pages for this forum project than there were for the other projects you have tackled. But the strategy that is used for this project is more or less the same for all pages. Here's what we'll be doing:   * A user clicks on a link on a forum page * The hyperlink will contain a variable that we can use to access our database tables * Take this code and pull records from the table   As an example, take the Microsoft Word Forum. You saw that the [hyperlink in the status bar](http://www.homeandlearn.co.uk/php/php16p3.html) is this, when you hold your mouse over it:  **pageThread.php?sID=secWP**  The page the user will be taken to is called **pageThread.php**. But we want to take some information to this page (which forum the user asked for). This is done by adding a question mark after pageThread.php. The question mark tells PHP that there is some GET data to follow. You then type a variable name (**sID**, for us). After an equals sign ( = ) you type what is going into the variable name (**secWP**, in the hyperlink above).  When **pageThread.php** is loaded, we can grab that **sID** variable and do something with it.  This, then, is the strategy we'll use for most of the forum – pass variables to other pages by using GET data. We need to do this because there is no easy way to hand variable information from one page to the next. You could write a cookie, or set up some session variables. But the technique we're using is a common one, and is widely used on the internet. There's a good chance you'll need to do this in your own projects, so it's well worth learning.  To begin the forum walkthrough, we'll take a look at the main page of the forum –**forumTest.php**. This is in the next part.  **The forumTest.php page**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p1.html)*.*    Start your server, and then load up the **forumTest.php** page in your browser by typing the following address (again, you can use localhost instead of 127.0.0.1):  **127.0.0.1/forum/forumTest.php**  You should see the main page of the forum (assuming that you've downloaded the files and saved them to the correct place). This one:  [The Basic Forum](http://www.homeandlearn.co.uk/php/php17p2.html) (Opens in a new window 80K).  The HTML is, of course, not something you'd want on your own site - that table looks far too basic! But the code is what we're interested in. Here's the coding strategy:   1. Get the number of posts in each forum 2. Get the number of replies for each post 3. Get the Forum main topics 4. Build up some hyperlinks 5. Print out a HTML table   The problem here is that we need to gain access to more than one table in our database. The Forum main topics are in the table called **forumsections**, the posts are in the table called **wpposts**, and the replies are in the table called **wpreplies**.  But the SQL is not too difficult, because we only need a limited amount of information: how many forum sections there are, and what the link text should be; a count of how many posts there are in each forum section; and a count of how many replies there are in each forum section.  Bearing this in mind, open up the code for the **forumTest.php** page.  At first sight, it looks rather long and daunting. You'll see some comments in the code, so as to break it down into manageable chunks.  The first eight lines of the code are just variables to hold the table HTML. This sets the look and feel of the forum. In fact, you can make a start right away in changing this!  **Exercise**  Even if your table HTML skills are a bit patchy, you can still change the colour scheme. On line 7 of the code, you'll see this:  **$tableHeaders = "<TR WIDTH = 200 height = 10 align = center  valign = middle bgcolor =#00EBEB>"**  Change the bgcolor to anything you like. Try these, and see what happens:  **#F84EFC #402C3E #74283E**  What happened with the darker colours? Try a few colours of your own, and see how you get on  Another thing you can change is the main cell colour. The cell colouring is done with this line:  **$tdStart = "<TD WIDTH = 200 height = 100 align = center valign = middle bgcolor =#F84EFC>";**  Again, change the bgcolor to anything you like. Start with the same colours as above.  Suppose you decided to go for one of the darker colours for the heading. How would you change the font colour for the heading text? Can you see what you would need to change? If you wanted white text, for example, the HTML would be this:  **<FONT COLOR = white>White Text</FONT>**  Where in the PHP would you put the FONT tag?    **Exercise**  Using the colours above, change your table so that it looks like this one:  [The Basic Forum](http://www.homeandlearn.co.uk/php/php17p2.html) (Opens in a new window 58K).   Back to the Code Ok, let's move on and examine the code.  After the table HTML, we set up some HTML for the hyperlinks:  **$hrefStart = "<A HREF = pageThread.php?sID"; $hrefEnd = "</A>";**  The important part here is the page we want to take the user to when a link is clicked, and the part after php. The page we want to take the user to is **pageThread.php**. But we're adding a question mark, and a variable name, as well: **?sID**. We'll put something into the **sID** variable later. This will be the GET data that we want hand to the page called pageThread.php. If you hold your mouse over each link, you'll see this sID variable change. It will be one of five values: **secWP**, **secXL**, **secVB**, **secWD**, and**secPH**. It's these values that we want to hand over to pageThread.php.  The next 15 lines in the code set up some arrays. We want five arrays. They are:  **$secIDs[] = array(); $tblPosts[] = array(); $tblReply[] = array(); $numPosts[] = array(); $numReply[] = array();**  You'll see what they all do as we go along. But notice the two arrays called $**tblPosts**[ ] and $**tblReply**[ ]. We've set these arrays up to hold the names of the tables for the posts, and the names of the tables for the replies:  **$tblPosts[] = "wpposts"; etc**  **$tblReply[] = "wpreplies"; etc**  The technique we'll use is to loop round these tables, and get information from each table.  The next six lines just get a connection to the database. You've met this code before, so we won't go into it.  An if statement comes next, to see if a connection to the database has been found:  **if ($db\_found) { }**  You can add an else part to this, if you like. This should say what to do if the database is not found. Print a simple "database not found" statement, for example.  The first code inside of the if statement is a for loop. We want to record how many posts there are in each forum section, so we need to loop round each table and count the number of rows in each post table. The for loop starts like this:  **for ($i = 1; $i < 6; $i++) { }**  The value of 6 is a hard-coded one. But this is not a good idea. Suppose you wanted to add more sections to the forum. The code would break right here. A better way to do this is use code to get how many rows there are in the forumsections table. You would then use this number in the for loop. For teaching purposes, though, we've stuck to a hard-coded value. We know there are only five main sections in our forum, and we won't be adding any more!  To count how many rows there are in each of the post tables, the first line of the for loop is this:  **$SQL = "SELECT \* FROM " . $tblPosts[$i];**  We need some SQL to hand to PHP, and this is what the line does. The first value we stored in the $tblPosts array was "wpposts". So the first time round the loop, the SQL variable will hold this:  **$SQL = "SELECT \* FROM wpposts"**  The next time round the loop, the SQL will be this:  **$SQL = "SELECT \* FROM wdposts"**  Each time round the loop, the only thing that changes about the SQL is the name of the table.  To count the number of rows in each table, we have this:  **$result = mysql\_query($SQL);**  **if ($result) {**  **$num\_rows = mysql\_num\_rows($result);**  **$numPosts[$i] = $num\_rows;**  **}**  We pass the SQL to the inbuilt PHP function **mysql\_query**( ). If any results are found then the variable called **$result** will be true. We're testing for this in the if statement.  If it is true, then next two lines will be executed:  **$num\_rows = mysql\_num\_rows($result);**  **$numPosts[$i] = $num\_rows;**  The first one just returns how many rows there are in a table. The second line puts the number of rows into the array we set up earlier - **$numPosts**. When we come to write our HTML for the hyperlinks, we'll use this array to print out how many posts there are in each forum section.  The next few lines of the code are for the replies. We want to record how many replies there are in each forum section. We do this in exactly the same was as for the posts - just loop round the $tblReply array and execute some SQL. When the loop is finished, the number of replies in each section is held in the array called **$numReply**:  **$numReply[$i] = $num\_rows;**  The next part of the code is a little bit trickier. We want to get the forum main topics, and build up the links. We're doing both of those things inside of another loop, a while loop this time. Here's the code:  **$SQL = "SELECT \* FROM forumsections"; $result = mysql\_query($SQL); $loopCount = 1;**  **while ($db\_field = mysql\_fetch\_assoc($result)) {**  **$secIDs[$loopCount] = $hrefStart . "=" . $db\_field['sectionID'] . ">" .  $db\_field['sections'] . $hrefEnd;**  **$loopCount++; }**  The first two lines set up some SQL, and then execute that against the database. The third line sets up a loop counter. We use this to access a different slot in the **$secIDs**array. Then we have the while loop (actually, we should be testing to see if **$result** is true, just like we did with the other two loops):  **while ($db\_field =mysql\_fetch\_assoc($result)) { }**  The variable **$db\_field** will hold the array data that is brought back from mysql\_fetch\_assoc($result). This inbuilt PHP function, if you remember, returns an array. The array that is brought back is a row from our forumsections table. (There are only two columns in this table.) The function will bring back data in this format:  **Column\_Name => Data**  To access the data in the array, you can then do this:  **$data = $db\_field[Column\_Name];**  That's what the first line of the while loop does. It's a long line that spills over into two on these pages:  **$secIDs[$loopCount] = $hrefStart . "=" . $db\_field['sectionID'] . ">" . $db\_field['sections'] . $hrefEnd;**  The part before the equals sign is this:  **$secIDs[$loopCount]**  $secIDs is the name of one of the arrays we set up at the top of the page. We want to store the hyperlinks in this array. The hyperlink in **$hrefStart**was this:  **$hrefStart = "<A HREF = pageThread.php?sID";**  We're adding the sectionID after **?sID** in the hyperlink above:  **$hrefStart . "=" . $db\_field['sectionID']**  After this code executes, it would give you something like this:  **pageThread.php?sID=secWP**  To add the text for the hyperlink, we also have this in the while loop:  **">" . $db\_field['sections'] . $hrefEnd;**  The right pointy arrow ( > ) completes the first part of the "A Href" HTML code. The text for the link is then this:  **$db\_field['sections']**  Finally, we add the rest of the HTML code for a hyperlink:  **$hrefEnd**  The last thing we do in the while loop is to increment the loop counter:  **$loopCount++;**  And that's the while loop! Yes, it's quite difficult. But study it for a while, and you'll get there. Remember: all we want to do is to build up an array of hyperlinks. Each hyperlink will be in this format:  **<A HREF = pageThread.php?sID=secWP>**Link Text**</A>**  The last thing we do in the forumTest.php code is to print out the HTML table. Here it is:  **print "<CENTER>"; print $TableStart; print $tableHeaders;**  **for ($i = 1; $i < 6; $i++) {**  **print $RowStart;**  **print $tdStart . $secIDs[$i] . $tdEnd; print $tdStart . $numPosts[$i] . $tdEnd; print $tdStart . $numReply[$i] . $tdEnd;**  **print $RowEnd;**  **}**  **print $TableEnd; print "</CENTER>";**  The first line just prints out the HTML code to centre things (note the American spelling). Then we have our two variables we set up at the top of the code. This prints out the HTML for the start of a table, and prints out the formatted headings we set up (these include the colour changes you made earlier).  Next, we have a for loop. What the loop does is to print out table code. It prints out the row start tag (<TR>), and then some table data tags (<TD>). Inside of each table data tag is the data from our three arrays $secIDs, $numPosts, and $numReply.  The last line in the for loop prints out the row end tag (</TR>). After the loop has finished, we print out the HTML for table end, and end the centre tag.  And that completes the code for the main page of our forum, forumTest.php. Here's a summary of what we did:   * Set up some variables to hold HTML table information * Set up some arrays to hold information from the database tables * Set up an array to hold all the hyperlinks * Accessed the database, and returned the number of rows in the posts and replies tables * Returned the information about each forum section, and built up a hyperlink * Looped round and printed out the table, the hyperlinks, the number of posts, and the numbers of replies   **Exercise** Play around with the HTMl Table code, and see if you can improve things. If you know any CSS, you can try to add that too.  Next, we'll take a look at the code for that page printed out in all the hyperlinks:**pageThread.php**.  **The pageThread.php page**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p2.html)*.*    When the user clicks on a forum section, they are taken to the **pageThread.php** page. The page looks like this:  [Posts in a Forum Section](http://www.homeandlearn.co.uk/php/php17p3.html) (opens in a new window - 110K)  If you are logged in as a member, however, you'll won't see that text at the bottom. You'll see this:  Forum Section  Only members are allowed to post topics and reply to others. The alternative is to have an open forum where anyone can post and reply. But this opens the forum to abuse, especially from people who want to spam your forum with lots of hyperlinks advertising their own sites and wares.  The login part of the forum is more or less the same one you met in the first walkthrough. But if you open up the code for pageThread.php (in the forum folder), you'll see a variable near the top of the page called $nonMember. If you're not logged in, the following text is added to the variable:  **$nonMember = "YOU NEED TO BE LOGGED IN TO POST (MAKE SURE COOKIES ARE ENABLED IN YOUR BROWSER)";**  As an exercise, change this message to anything you like. Then reload the page to see your new message.  If you are logged in as a member, the $nonMember variable will be blank. Later in the code, an if statement will check what is inside of the $nonMember variable. If it's blank, print the link to post a topic; if it's not blank, print the message.  After the session variable is checked, we have this line:  **include 'forumHTML.php';**  Here, we're including a file that holds all the HTML for the table, and a hyperlink. Open this page up, and you'll see the same table code you met in the main forum. The reason it's in a page of it's own is just to cut down on the amount of code in the pageThread.php page. We could have done the same with the forumTest.php page, but we thought we'd show you both techniques!  **Exercise** Change the colours of the table headers for the Posts page, and the colours for the cells in the table. Change the text in the headers to any colour you like.  After the include file, we have a function called **getPostSQL**( ) and one called**getReplySQL**( ). These are used to get some SQL, and you'll see how they work soon. But after the function, we have this:  **if ($\_SERVER['REQUEST\_METHOD'] = = 'GET') { $secCode = ''; if (isset($\_GET['sID'])) { $secCode = $\_GET['sID']; } }**  Here, we're checking to see if the page in the browser was loaded via a hyperlink click or a refresh (GET). If it was, there will be some GET information stored that we can try to retrieve. We're setting up a variable called $secCode. This starts off a being blank:  **$secCode = '';**  But if our variable from the previous page is set (the sID variable), we can return this into the variable called $secCode. So $secCode is either going to be blank, or it will have one of our five values: **secWP**, **secXL**, **secVB**, **secWD**, or **secPH**.  The next line is an if statement:  **if ($secCode <> '') {**  **}**  Here, we're checking if the variable called $secCode is not blank. Only if it's not blank will the code for the if statement be executed.  If it is blank, then something went wrong. In which case, the code at the bottom gets executed:  **else { print "Forum Not Available"; }**  For the rest of the code, we need to do the following:   * Set up an array to hold all the posts * Find out how many replies there are for each post * Find out which member posted the thread * Print the table out * Print out the "login" link, or the "Non Member" information   There's quite a lot to do! Again, though, if you open up the code for **pageThread.php**you'll see a lot of comments, breaking it down into manageable chunks.  The first four lines to examine are these (the third line spills over into two, on these pages):  **$postData[] = array();**  **$replyHTML = "?sid=" . $secCode; $replyHTML = "<A HREF = postForm.php" . $replyHTML . ">Create a new post</A>";**  **$forum = $secCode;**  We set up an array called **$postData**[]. This will hold the information about each post. The next two lines set up the "Create a new post" hyperlink. If the member wants to create a new post, you need to pass in which forum section they want to add the post for. Was it the Word section, the Excel section, the VB section, etc? The only thing you need here is whatever is inside of the $secCode variable (secWP, secXL, etc).  The fourth line just passes whatever is inside of the $secCode variable to a new variable called $forum.  The next six lines connect to the database. After that, we have an if statement to check if the database was found. This is the same code as before. Set up an array to hold all the posts Inside of the**$db\_found** if statement, we have our first bit of code from our list: Set up an array to hold all the posts.  The first line is this:  **$secCode = getPostSQL($secCode);**  This is a call to one of our functions at the top of the page. We're handing the function whatever is inside of the secCode variable. The result of the function will be returned to the same variable (**$secCode =** ).  If you study the function, you should be able to figure out what it does. But it's just a series of if statements that check what is inside of $sectionCode. Is it secWP, secWD, secVB, secXL or secPH?  Whichever one it is, some SQL gets returned:  **$sql = "SELECT \* FROM wpposts ORDER BY datePosted DESC";**  The only thing that changes is which table we want to pull data from. But notice the ORDER BY clause at the end. This is another SQL command you can use. It sorts results based on a table column that you provide. Here, we want to sort the results based on the datePosted column. We want a descending sort, so we've used DESC. If you want an ascending sort, use ASC.  **Exercise**  Change DESC to ASC. Save the page, and reload it in your browser. Watch what happens.  After we grab the SQL, we hand it to PHP for processing:  **$result = mysql\_query($secCode);**  If any results are returned, the $result variable will be true. After setting a rows counter to zero, we get the number of rows in a fourm section:  **$totalRows = 0; $totalRows = mysql\_num\_rows($result);**  Next, we check how many rows were returned:  **if ($totalRows <> 0) {**  **} elseif ($totalRows = = 0) { print "This Forum is not yet available"; }**  If the total number of rows is not zero, then the main code is executed. If no rows were returned, we can print out an error message.  The first thing to get executed inside of the new if statement is a for loop:  **for ($i = 0; $i < $totalRows; ++$i) { $postData[$i] = mysql\_fetch\_array($result); }**  What we're doing here is looping round each row in a database table, and putting the results into an array called **$postData**. But this is a new type of array - something called a 2D array. As this page is getting a bit too long, we'll continue the lesson in the next part.  **2D Arrays and PHP**   |  |  |  | | --- | --- | --- | | *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p3.html)*.*    In a normal array, each position can hold only one value. Like this:  **$My\_Array[] = array();**  **$My\_Array[0] = 10; $My\_Array[1] = 20; $My\_Array[2] = 30;**  So position 0 in the array above holds a value of 10, position 1 holds a value of 20, and position 2 holds a value of 30.  But for us, we're returning a row of data from our table. Each row will be like this: | | | |  | **Column Name** | **Value** | |  | threadID | pos3 | |  | memberID | mem1 | |  | threadTopic | Clip Art | |  | postText | Got some good clip art? | |  | datePosted | 2006-04-13 12:11:06 | | In the post tables, there are 5 columns. When we use mysql\_fetch\_array( ) it will fetch back each column name, and it's value. You can store all of this information into an array of your own. But now, each position in the array will hold more than one value. It will hold a row of data from the database table. The new array will be like this:  **$My\_Array[0] = "pos3", "mem1", "Clip Art", "Got some good clip art?", "2006-04-13 12:11:06"**  So position 0 now holds more than one value. It holds 5 values, in the code above. This is a 2D array - an array where each postion holds more than one value. (You can also have a 3D array, but this is far more complex than we need.)  If you want to just access the datePosted value, you can do it like this:  **$postData[0]['datePosted']**  So the name of you 2D array goes first, followed by square brackets. In between the square brackets, you need a postion in the array. To access just a particular value in that postion, you type a column name (or key value). In the line above, we've specified the datePosted column.  If all that is a bit confusing, try this exercise.  **Exercise** Add the following to the end of your for loop (the new line is in red):  **for ($i = 0; $i < $totalRows; ++$i) { $postData[$i] = mysql\_fetch\_array($result); }**  **print $postData[0]['threadTopic'] . "<BR>";**  Refresh your page and see what happens. Now change the 0 to 1, save your work, and refresh the page. Now change 'threadTopic' to 'datePosted'. Again, reload the page. Try the other Column Names from the table above.  You can also add new keys and values to a 2D array. Simply type a new name in between the square brackets, and its value. Like this:  **$postData[0]['newValue'] = "new value here";**  Exercise Add this new for loop to your code (the new lines are in red):  **print $postData[0]['threadTopic'] . "<BR>";**  **for ($i = 0; $i < $totalRows; ++$i) { $postData[$i]['newValue'] = $i; }**  **print $postData[0]['newValue'] . "<BR>";**  Save your work, refresh the page, and watch what happens. Now change the 0 to 1, save your work, and refresh the page. What did you notice?  In summary: use a 2D array when you want each position in your array to hold more than one value.  If you're still having problems grasping the concept of 2D arrays then think of them like an Excel spreadsheet. Each row in the spreadsheet represents a postion in the 2D array. Each column in the spreadsheet represents a value for each postion. A normal array would look like this:  A normal Array  And a 2D array would look like this:  A 2D Array    And now on with the forum code, which we'll continue in the next part. (Delete any code you added for the exercises above.) | | |   **Find out how many replies there are for each post**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p4.html)*.*    We've just used a 2D array to store the row information from our database table. Each postion in the 2D array will hold the threadID, the memberID, the threadTopic, the postText and the datePosted. The next thing to do is find out how many replies there are for each post. The code that does that is as follows:  **$cnt = count($postData);**  **for ($i = 0; $i < $cnt; ++$i) { $rep = $postData[$i]['threadID'];**  **$repSQL = getReplySQL($forum, $rep); $result = mysql\_query($repSQL); $numRows = mysql\_num\_rows($result);**  **$postData[$i]['numRows'] = $numRows; }**  First, we get a count of the number of positions in the 2D array. This is so that we can loop through each postion. The first line in the for loop is this:  **$rep = $postData[$i]['threadID'];**  This returns the threadID from each position. We're using the variable $i to access each postion in the array.  Next, we make a call the other function at the top of the code:  **$repSQL = getReplySQL($forum, $rep);**  The second function has two arguments, a section code ID (now stored in the ($forum variable), and the threadID from the array. The function is again a series of if statement. Depending on which forum section ID is being passed over, a SQL statement is returned. Look at the SQL, though:  **"SELECT \* from wpreplies WHERE wpreplies.threadID = '$reply'";**  We're saying, Select all the records from the wpreplies table where there is a match on the threadID". The value in $reply is coming from our 2D array, and will be something like pos1, pos2, po3, etc. This is from the Posts table. Because of the way we set up our database, we have a threadID field in both the replies and post tables. This is what allows us to link both tables in the above SQL.  After the SQL executes, it will return the number of rows where the two threadID's are the same. This number is then stored into the variable called $numRows. The final line of the for loop is this:  **$postData[$i]['numRows'] = $numRows;**  Here's we're adding a new key and a new value to our 2D array. The key is between the second set of square brackets, and is called **numRows**. The value for this key is whatever is inside of the variable called **$numRows**. This allows us to store a record of how many replies there are in each posts.  In the next part, you'll see how to find out which member posted.  **Find out which member posted**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p5.html)*.*    The next thing we need to do is find out which member posted the original thread. The code in the **pageThread.php** script that does this is as follows:  **for ($i = 0; $i < $cnt; ++$i) { $memb = $postData[$i]['memberID'];**  **$memSQL = "SELECT \* from members WHERE memberID = '$memb'";**  **$result2 = mysql\_query($memSQL);**  **if ($result2) { $db\_field = mysql\_fetch\_assoc($result2); $memName = $db\_field['username']; $postData[$i]['member'] = $memName; } }**  The for loop uses the same variable that we set up to count the number of positions in the 2D array. We're looping round each position in the 2D array and grabbing the memberID. The line that does that is this:  **$memb = $postData[$i]['memberID'];**  When we set up the 2D arrray, one of the values that was returned from the table was the memberID. We're storing this in a variable called $memb.  The next line is this:  **$memSQL = "SELECT \* from members WHERE memberID = '$memb'";**  We're setting up some SQL here. Because we set up our database with a memberID in three of the tables, we can select all the records where the memberIDs match. Again, this shows you the benefits of planning your database before you start!  The next few lines are these:  **$result2 = mysql\_query($memSQL);**  **if ($result2) { $db\_field = mysql\_fetch\_assoc($result2); $memName = $db\_field['username']; $postData[$i]['member'] = $memName; }**  First, we run the SQL:  **$result2 = mysql\_query($memSQL);**  Next, we have an if statement testing to see if the variable called $result2 is true. If it is, the next few lines are executed. (We should really have an else part here, as well. This should say what happens if $result2 is false.)  The first line inside of the if statement is this:  **$db\_field = mysql\_fetch\_assoc($result2);**  This brings back the result as an array. The array will return all the columns and the data from the table row. This is then placed into the variable called **$db\_field**.  The only thing we need is the username. So the line is this:  **$memName = $db\_field['username'];**  We're placing the username in a variable called $memName. This is then added to the 2D array:  **$postData[$i]['member'] = $memName;**  We now have a new key and value in our 2D array - member. This holds the member's username. If you wanted to display other information about the member, you would do it here. Get the email address, for example, or that Avatar we didn't set up! You'd do it like this:  **$email = $db\_field['username']; $postData[$i]['emailAddress'] = $email;**  In other words, grab the column name from the table, and add a new key to the 2D array.  In the next part, you'll see how to print out all the values we've grabbed from the**pageThread.php** page.  **Print the table out**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p6.html)*.*    The next thing to do is to print out all the information we stored in our [2D array](http://www.homeandlearn.co.uk/php/php17p4.html). We're printing it out in a table.  The technique is the same as the one in for the forumTest.php page - just print out your HTML table headers, and loop round for the table rows and table data tags. Here's just one line from the loop:  **print $tdStart . $postData[$i]['member'] . $tdEnd;**  We print out the TD start tag, and the TD end tag. In between that, we have this:  **$postData[$i]['member']**  To print out a value from the 2D array, just refer to its position in the array, and the key you want to print. The position is coming from the loop ($i). The key is member.  Examine the rest of the for loop that prints out the table. See if you can understand what's going on. Especially this rather long line (split into three lines on these pages):  **print $tdStart . $hrefStart . "=" . $postData[$i]['threadID'] . "&forum=" . $forum . "&pageID=0" . ">" . $postData[$i]['threadTopic'] . $hrefEnd . $tdEnd;**  An important part of the line above is this:  **"&pageID=0"**  As you may have realised, the whole line prints out the hyperlink. If you hold your mouse over a hyperlink on the pageThread.php page, you'll see something like this:  **pageReply.php?rID=1&forum=secWP&pageID=0**  This means that we are trying to pass three things to a page called pageReply.php. We're trying to pass the following three variables:  **rID forum pageID**  The values in the variables are **1**, **secWP**, and **0**. You'll see how they work in the next section. But the pageID is used to display the links that will take a user to say page 1 of the replies, or page 2 of the replies (if there is a page 2), page 3, page 4, etc. We're passing a value of zero because this is the first page of the replies.  The final part of the pageThread.php code is this:  **if ($nonMember = = '') {**  **print "<P align = center>" . $replyHTML . "</P>";**  **} else {**  **print "<P align = center>" . $nonMember . "</P>"; print "<P align = center>" . "<A HREF = login.php>Login Here</A>" . "</P>";**  **}**  This just tests what is inside of the variable we set up at the top of the page. If $nonMember is blank, then we can print out the hyperlink to allow the member to post a new topic. If it's not blank, then we can display some HTML asking the user to login in. You can also add a link to register, if you like. We did this in a previous section, so we won't cover it here.  But that's it! That's the code to display all the posts in your forum.  There is however, a problem. Supppose your forum is really large. If so, you'd need to spread the posts over more than one page. That way a user can click onto page 2 of the posts, page 3, etc. You'll see how to do this in the next section, which explains how to write the code for displaying the replies to a post - we'll definitely be needing that**pageID**!  **The pageReply.php Code**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p7.html)*.*    The code for the **pageReply.php** page is a little bit longer than the others. This page has to do the following:   * Check if the user is a member. If so, display a link so they can post a reply. * Set up some functions to handle the SQL * GET the post and the forum section that the user is replying to * GET the pageID just in case there is more than 1 page of replies, allowing us to set hyperlinks for each page * Display information about the original post * Display which member posted * Find out which members replied * Display all the replies, but limit them to ten replies per page   When a user clicks on a Post, they will see this page:  [Reply to a Post](http://www.homeandlearn.co.uk/php/php17p8.html) (opens in a new window 125K)  This is the page we're now going to examine.  The code that checks if the user is a member or not is the same code for the previous section. After the functions, though, we need to GET the data that was passed by the previous page. This is the data in the **rID** variable, the forum variable, and **pageID**variable:  **if ($\_SERVER['REQUEST\_METHOD'] = = 'GET') { $secCode = ''; $postID = '';**  **if (isset($\_GET['rID'])) { $postID = $\_GET['rID']; $secCode = $\_GET['forum']; $pageID = $\_GET['pageID']; } }**  Here, we're checking to see if the page was loaded using the GET method. If it was, we first set a variable called $secCode and a variable called $postID to blank strings. We then test to see if the rID variable has been set. If so, we can get the information that was handed by the previous pages:  **$postID = $\_GET['rID']; $secCode = $\_GET['forum']; $pageID = $\_GET['pageID'];**  The code after that, you've already met in the previous section: Test to see if secCode is blank, build up some HTML for the reply link, and open a connection to the database.  If the database is found, we first need to find out how many people relied to a Post:  **$repCode = getReplySQL($secCode); $repCode = $repCode . "'" . $postID . "'";**  **$result = mysql\_query($repCode); $totalRows = mysql\_num\_rows($result);**  First, we make a call to one of the functions at the top of the page. This returns some SQL. The second line adds the postID to the SQL. When the SLQ is executed on the third line, we'll have all the replies for a particular post. The forth line tells us how many replies there are in total. We're putting this into a variable called **$totalRows**.  If the **$totalRows** is not zero, then we can get some more SQL. Since we're only going to be displaying ten links per pages, we can limit the data we pull from the replies table. After all, there's no sense in getting all the replies, if we're not going to be displaying all of them. To limit the number of replies, we have this:  **$repCode = getReplySQL($secCode); $repCode = $repCode . "'" . $postID . "'" . " LIMIT " . $pageID . ", 10";**  The first line again makes a call to the functions at the top of the page. This gets some SQL that we can use against the reply tables.  The second line is where we're building up some SQL to limit the number of replies that will be pulled from the table. The first part of the second line just adds the **$postID** to the SQL:  **$repCode = $repCode . "'" . $postID . "'"**  The rest of the line is where we're limiting the results:  **LIMIT " . $pageID . ", 10";**  The SQL command LIMIT, as its name suggests, allows you to limit the number of results that you pull from a table. Take this SQL as an example:  **SELECT \* FROM tblReply LIMIT 0 10**  The first number after LIMIT is which record from the table you want to start at. The number zero means start pulling data from the first record in the table. The second number says how many records you want to grab.  In our code, the start value is coming from the variable called **$pageID**. This is passed to the code when a hyperlink is clicked. It started off as zero, if you remember, because this will be the first page of results. If we wanted to start pulling data from row number 11, as we do if there is more than 10 replies, then we need to change the value in the variable called **$pageID**. This values gets changed later, as you'll see. You can then add it to the hyperlinks.  We can then execute the SQL, and put the data from the table into an array:  **$result = mysql\_query($repCode); $numRows = mysql\_num\_rows($result);**  **if ($result) { for ($i = 0; $i < $numRows; ++$i) { $replyData[$i] = mysql\_fetch\_array($result); } }**  Notice how the loop goes from 0 to **$numRows**. Inside of the loop, we're putting each row from the table into an array called $replyData.  But we also need to find out how many links are needed. If there are 11 replies to a post, we need 2 links. Clicking the second link will take you to the second page of replies. If we had 21 replies, we'd need three links - 10 on the first 2 pages, and 1 on the third. The code that counts how many links are needed is this:  **$cnt = count($replyData); $linkNum = floor($totalRows / 10);**  The first line just counts how many replies are in the array. We pass this to a variable called $cnt for use later in the code.  The next line divides the number of rows by 10. We use the floor( ) function to get rid of any "point something" at the end. 11 divided by 10, for example, would give us 1.1. We don't need the .1 at the end, so floor( ) will strip this off.  The next bit of code gets information about the thread:  **$result = mysql\_query($posCode); $numRows = mysql\_num\_rows($result);**  **if ($numRows = = 1) {**  **$db\_field = mysql\_fetch\_assoc($result); $topic = $db\_field['threadTopic']; $postText = $db\_field['postText']; $datePosted = $db\_field['datePosted']; $memPost = $db\_field['memberID'];**  **}**  We first execute the SQL we set up at the top of the code. This returns the original Post. But we need to check if a row is returned. If a row is returned then we fetch it back as an array:  **$db\_field = mysql\_fetch\_assoc($result);**  The variable $db\_field will then hold the fields from the Posts table. The next four lines return the threadTopic, the posText, the datePosted, and the memberID. These are all fields in the post tables. We then put theses into variables of their own:  **$db\_field = mysql\_fetch\_assoc($result); $topic = $db\_field['threadTopic']; $postText = $db\_field['postText']; $datePosted = $db\_field['datePosted']; $memPost = $db\_field['memberID'];**  We can then use these variables later in the code.  To find out which member posted the thread, we have this:  **$memSQL = "SELECT \* from members WHERE memberID = '$memPost'"; $result = mysql\_query($memSQL);**  **if ($result) { $db\_field = mysql\_fetch\_assoc($result); $postName = $db\_field['username']; }**  This is the same code you met in the previous section. We use SQL to pull the record from the members table WHERE there is a match on the memberID fields. This is then placed into a variable called $postName.  To find out which members replied to the post, we have this code:  **for ($i = 0; $i < $cnt; ++$i) {**  **$memb = $replyData[$i]['memberID']; $memSQL = "SELECT \* from members WHERE memberID = '$memb'"; $result2 = mysql\_query($memSQL);**  **if ($result2) { $db\_field = mysql\_fetch\_assoc($result2); $memName = $db\_field['username']; $replyData[$i]['member'] = $memName; } }**  What we're doing here is looping round the **$replyData** array. This is the array we set up earlier that holds information about the replies. We just want the memberID. We then execute some SQL on the members table WHERE there is a match on the memberID field. Once we find a match, we add a new key to the $replyData array:  **$memName = $db\_field['username']; $replyData[$i]['member'] = $memName;**  The next thing we do is to print the links out, so that users can go to other pages. As a reminder, here's the links we want to print out:  The forum links  Hold your mouse over these links and you'll see this in the status bar:  The status bar for page 2  Or this, for Page 1:  The status bar for page 1  The only thing to change here is the **pageID**. If you click Page 1, **pageID** is zero. This is because you want to pull records 0 to 10 from the replies. For page 2, **pageID** is 10. You can then use this number to pull records number 11 onwards. Here's the code that does that:  **$linkCount = 0; $pageCount = 1;**  **for ($i = 0; $i <= $linkNum; ++$i) {**  **$linkPages = "<A HREF = pageReply.php?rID=" . $postID . "&forum=" . $secCode;**  **$linkPages = $linkPages . "&pageID=" . $linkCount . ">Page " . $pageCount . "</A>";**  **print $linkPages . " "; $linkCount = $linkCount + 10; $pageCount++;**  **}**  The variable $linkNum hold the number of pages that are needed. What we're doing here is looping round and creating links. The links are printed out with: print $linkPages . " ";  But the two lines above this are where the hyperlinks get built. Study the code and see if you can work out how it works. Compare the address in the status bar, with the code in the loop. Here's the status bar (concentrate on everything after pageReply.php):  And here's the two lines in the code that builds the link:  **$linkPages = "<A HREF = pageReply.php?rID=" . $postID . "&forum=" . $secCode;**  **$linkPages = $linkPages . "&pageID=" . $linkCount . ">Page " . $pageCount . "</A>";**  Finally, we can then print out the HTML table, using all the information we have gathered so far. Again, study the code. You should, by now, be able to figure out what's going on. But we're just looping round printing table data.  The only thing that remains now is to look at the code for posting a reply, and for adding a new post to the forum.  We'll do that next.  **The Reply Form**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p8.html)*.*    In the last section, there was a variable called **$replyHTML**. This is a hyperlink that, when clicked on, leads to a page called replyForm.php. Here's the code:  **$replyHTML = "?pid=" . $postID . "&sec=" . $secCode;**  **$replyHTML = "<A HREF = replyForm.php" . $replyHTML . ">Reply to this post</A>";**  When the link is displayed, and the mouse held over the link, you'll see this in the status bar at the bottom (you'll only see this link of you're logged in as a member):  Reply hyperlink  The code above for the $replyHTML variable is adding the following after the question mark:  **pid=pos1 sec = secWP**  When the link is clicked, we're passing these two variables to the page called replyForm.php. The pos1 is which post the member wants to reply to, while the secWP is the forum section for Word Processing. When the member types the reply, we'll use these values to update the database table.  So, open up the code for replyForm.php (in your forum folder), and we'll see how it works.  (If you display the page in a browser, you'll see it's just a text area and a button. The member types the reply, and clicks the button.)  The first bit of code is just the usual checks to see if the user accessing the page has logged in. If they have, we grab the memberID, and put it in a variable:  **session\_start(); if (!(isset($\_SESSION['login']) && $\_SESSION['login'] != '')) { header ("Location: login.php"); } else { $memberid = $\_SESSION['memID']; }**  We also need to grab those variables from the previous page:  **if ($\_SERVER['REQUEST\_METHOD'] = = 'GET') { $secCode = ''; $postID = ''; if (isset($\_GET['pid'])) { $postID = $\_GET['pid']; $secCode = $\_GET['sec']; } }**  So the pid variable (the one that contained "pos1", from the previous page) is handed over to a variable called $postID:  **$postID = $\_GET['pid'];**  And the sec variable (the one that contained secWP) is handed to a variable called $secCode:  **$secCode = $\_GET['sec'];**  All the PHP code does is to print out HTML for a FORM. The form will contain a textarea and a button. When the button is clicked, we'll process the data on another php page. The ACTION attribue of the FORM tag is where the processing page is located:  **… METHOD ='POST' ACTION ='results.php'>";**  We're going to be using the POST method to hand over our values to a page called results.php. Notice that the FORM also has hidden values:  **$hidSec = "<INPUT TYPE = Hidden Name = h1 VALUE =" . $secCode . ">"; $hidPost = "<INPUT TYPE = Hidden Name = h2 VALUE =" . $postID . ">"; $hidMem = "<INPUT TYPE = Hidden Name = h3 VALUE =" . $memberid . ">";**  And there's our three variables: **$secCode**, **$postID**, and **$memberid.** When the button on the form is clicked, these hidden variables will get handed over to the page that processes the data - the results.php page.  Notice, too, that the NAME attributes for these hidden variables are **h1**, **h2**, and **h3**. The NAME of the textarea on the form is post. We'll be passing all these values to the processing page ( **results.php**).  We'll have a look at that page in the next part.  **The results.php page**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p9.html)*.*    There's not too much code in the **results.php** page. All we need to do here is to INSERT a reply to a post into the correct database table. Open up the code, and we'll take a look at it.  The first thing to notice is the function at the top of the page – **getReplySQL**( ). This takes one argument – the section code for the forum.  After the function, we check to see if the form was submitted using the POST method:  **if ($\_SERVER['REQUEST\_METHOD'] = = 'POST') {**  **}**  If this returns a value of true, then we grab that data from the form:  **$secID = $\_POST['h1']; $posID = $\_POST['h2']; $memID = $\_POST['h3']; $repText = $\_POST['post'];**  We're grabbing all those HIDDEN values from the form (which had the NAMES **h1**, **h2**, and **h3**), as well as the text in the textarea (which had the NAME post).  The next three lines just add some single quotes. This is NOT how you'd want to do it in your own forum! There are security issues here, because you're grabbing text from a textarea and trying to INSERT it into a database. We covered the security issues in an earlier section, but try this exercise:  **Exercise** Login to the forum with one of the username/passwords you saw earlier (these are in the textfile called **uandp.txt**, in your **forum** folder). Reply to the post "A Brand New Forum". You'll see this basic form:  A Basic Reply Form  If you just go ahead and click the button, the text **Some text here** should get added to the database.  But surround the text with single quotes. Type this:  **'Some text here'**  Now click the button. You should see the message "**No results**". This is because the code couldn't execute the SQL with those single quotes added. How would you solve this? Review the section on security, especially the parts about SQL injection attacks. Wasn't there something about magic quotes, and strip slashes? How would you use these to lessen a SQL injection attack?  Another secrurity issue to worry about is how much text to allow the user to type into the textarea. You can check how long a string is with the strlen( ) function. A good run-down on how to use this can be found in the PHP manual here:  <http://uk.php.net/manual/en/function.strlen.php>  A simple way to use this would be:  **$strCount = strlen($repText);**  **if ($strCount > 255) { print "too many characters in your reply"; } else { //Do the rest of the code here }**  This just prints an error message if the user types in more than 255 characters. (However, we've set the reply text field in the database tables to MEDIUMTEXT. This can hold far more characters than 255. If you're restricting users to 255 characters, then TINYTEXT would be a better option.)  But back to our (less than secure) code. We've just grabbed the data from the FORM and placed it into variables:  **$secID = $\_POST['h1']; $posID = $\_POST['h2']; $memID = $\_POST['h3']; $repText = $\_POST['post'];**  The secID will contain something like "secWP", the posID will contain something like "pos1", and the member ID is always "mem" plus a number (mem1, me2, etc). The $repText variable holds the reply the user typed in the textarea.  The next code we have is some date code:  **$date\_today = date("Y-m-d H:i:s"); $date\_today = "'" . $date\_today . "'";**  The characters we're using between the round brackets of the date function mean we want the year first, then the month, then the day. These will be separated with the "-" character. We're also adding a time part that returns Hours, Minutes, and Seconds. The second line adds single quotes to the date, so we can use it in the SQL.  The next line is a call to the function at the top of the page:  **$tableSQL = getReplySQL($secID);**  The function returns SQL to be used against the database table. But it only returns part of the SQL we need:  **$sql = "INSERT INTO wpreplies (threadID, memberID, reply, dateReplied) VALUES ";**  In between the round brackets, we have the fields from the table. But we don't yet have the VALUES we need to insert into these fileds. The next line in our code adds the values. It's a long line that spills over into two on these pages:  **$tableSQL = $tableSQL . "(" . $posID . "," . $memID . "," . $repText . "," . $date\_today . ")";**  We're building up the $tableSQL string variable here. It already holds the first part of the SQL we need. We're now adding the VALUES. We need the round brackets, the data from the variables, and the commas. After the line is executed, the string would be something like this:  **"INSERT INTO wpreplies (threadID, memberID, reply, dateReplied)VALUES (pos1, mem1, "My Reply", "2006-10-22 13:30:14")"**  After we open a connection to the database, we try to execute the query:  **$result = mysql\_query($tableSQL);**  Then we check to see if the SQL is executed successfully:  **if ($result) { print "Your Reply has been added to the Forum." . "<BR>"; print "<A HREF = forumTest.php>Back to the forum</A>" . "<BR>"; } else { print "no results" . "<BR>"; }**  And that's all we need to do on the this page: try to insert the reply into a reply table in the database. Some of the error checking has been left out, so as not to confuse the main techniques used. But you should implement the error checking in your own code. Especially the checks to ward off attacks on your database!  In the next part, we'll see how to write the code to post a Topic on the Forum  **Posting a Topic on the Forum**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p10.html)*.*    The final part of the walkthrough looks at how to allow users to post a new topic on the forum. There is one useful database technique that may come in handy in your own code - how to deal with primary key fields that are not auto-incrementing numbers. Let's make a start.  The Post Form The form that the user fills in to post a new topic is this:  A Basic form to Post forum replies  As you can see, it's fairly basic, and you can definitely improve on this!  The form is similar to the Reply form, except for the addition of a text box. And we're doing the same things in the code: hand over values to another php page for processing. If you open the page called **postForm.php**, you can examine the code for yourself.  Again, we're using the POST method to post the form data to a php page:  **METHOD ='POST' ACTION ='resultsP.php'**  The page we're posting the data to is called resultsP.php. We'll take a look at this code in a moment, but notice the hidden variables:  **$hidSec = "<INPUT TYPE = Hidden Name = h1 VALUE =" . $secCode. ">"; $hidMem = "<INPUT TYPE = Hidden Name = h2 VALUE =" .$memberid . ">";**  We want to hand over the forum section code (secWP), and the member id. The text box on the form has been given the name **tp**, and the text area is called **post**. With this in mind, take a look at the **resultsP.php** page (in your **forum** folder).  **The resultsP.php page**  *This lesson is part of an ongoing Foruml tutorial. The first part is here:*[*Build your own Forum*](http://www.homeandlearn.co.uk/php/php16p1.html)*, along with all the files you need. The previous lesson is*[*here*](http://www.homeandlearn.co.uk/php/php17p11.html)*.*    At the top of the code for the **resultsP.php** page, there's two functions. We'll take a look at these later. But bear in mind what this page does – inserts the new post into the database table. (The security issues discussed in the previous section apply here, as well)  To get the data posted from the FORM, we have this:  **$secID = $\_POST['h1']; $memID = $\_POST['h2']; $posTopic = $\_POST['tp']; $posText = $\_POST['post'];**  This is the same technique we used in the previous section: just put the POST data into variables. The **$secID** variable will hold something like secWP, the **$memID** variable will hold the member id, the **$posTopic** variable will hold the heading for the topic, and the **$posText** variable will hold the text of the post itself.  After we get the connection to the database, we run in to our first problem. The problem is that the Primary Keys in the post tables are not auto-incrementing numbers. With an auto-incrementing number, you can usually leave the database to update this field –all it needs to do is to add 1 to the previous value. Like this:  **ID 0 1 2 3**  If you update the table, the database would automatically add 1 to the ID field, and the next row would be 4. You don't have to do anything.  But for the Primary Key in the post tables, we have a field called **threadID**. The threadID field looks like this:  **threadID pos1 pos2 pos3**  So you can't just add 1 to this field, if you insert a new row. You have to make sure that the new row is pos4, and the next new row will be pos5, etc.  There's another problem as well. How can you be sure that the database hasn't inserted your rows like this (and it will!):  **threadID pos1 pos3 pos2**  So the last row in the table is pos2. If you try to update this with pos3 as the new row, you'll get an error. Because there already is a pos3. A Primary Key field is one that has unique values. And that why you'd get an error.  This is a common problem when you have your own format as the Primary Key in a database table – adding a new unique value when that value is not an auto-incrementing number.  In the example above, we need make sure that the new value in the threadID field is pos4. This is a run-down on how we'll do it:   1. Get all the posts from the table 2. Set up an array to hold the threadID data 3. Strip the "pos" part, and just leave the number 4. Sort the array with the lowest number first and the highest last 5. Get the last value in the array (which will be the highest number) 6. Add 1 to this number 7. Put the "pos" part back 8. Update the threadID array   If you open up the code for the resultsP.php page, you'll see comments that tackle the items in list above. The first part of the code gets all the posts from the table:  **$SQL = getPostTable($secID);**  This is just a call to one of the functions at the top of the page. When the function is run, you'll have SQL like this:  **"SELECT \* FROM wpposts";**  This selects all the records from a table called wpposts. The next two lines are these:  **$result = mysql\_query($SQL); $numRows = mysql\_num\_rows($result);**  The first line executes the SQL. The second line returns how many rows we have returned from table. We'll need this for the for loop. We then set up an array to hold the threadID values:  **$posNums = array();**  We now need to loop round the rows in the table, and get the threadID field:  **for ($i = 0; $i < $numRows; $i++) {**  **$row = mysql\_fetch\_row($result); $pID = $row[0];**  **$posNums[$i] = ltrim($pID, 'pos');**  **}**  The loop goes from zero, to less than $numRows. The first line in the for loop is this:  **$row = mysql\_fetch\_row($result);**  The inbuilt function mysql\_fetch\_row, as its name suggest, fetches a row from a table. It will fetch it back with row[0] as the first table column, row[1] as the second column, row[2] as the third, etc. Because we know that row[0] is our threadID field, we can grab this value:  **$pID = $row[0];**  The treadID , then, is placed into a variable called $pID. To strip off the "pos" part of the threadID, we have this:  **$posNums[$i] = ltrim($pID, 'pos');**  We're using the unbuilt function ltrim( ) to trim the "pos" part. In between the round brackets, you type the text you want examine. After a comma, you type the text you want trimming.  After the loop finishes, we might end up with something like this:  **$posNums[] 1 3 2**  So we have all the numbers in an array, but they are not sorted from lowest to highest. The next line does that:  **sort($posNums);**  The inbuilt function sort( ) is used to sort the array. The name of the array you want to sort goes between the round brackets of the function. But note that the square brackets of the array go missing.  Once the array is sorted, we can get the last item in the array:  **$lastID = end($posNums);**  Again, we're using an unbuilt function. The end( ) function is used to move to the last element of an array. Here, we're putting the value returned by the function into a variable called $lastID.  Once we have the last number, increment it:  **$lastID++;**  Finally, once we have the last number, we can add the "pos" back on:  **$threadid = 'pos' . $lastID;**  All that coding gets us the highest pos number from our threadID field. We can then go ahead an get the SQL for our INSERT statement:  **$tableSQL = getPostSQL($secID);**  This calls the function at the top of the page. The function will return something like this:  **$sql = "INSERT INTO wpposts(threadID, memberID, threadTopic, postText, datePosted) VALUES "**  We want to INSERT INTO the wpposts table a set of VALUES. The names of the table columns go between the round brackets. But we still need to add the VALUES. This is done with the rather long next line:  **$tableSQL = $tableSQL . "(" . $threadid . "," . $memID . "," . $posTopic . "," .$posText . "," . $date\_today . ")";**  We're just building up a string. Something like this:  **INSERT INTO wpposts(threadID, memberID, threadTopic, postText, datePosted) VALUES (pos1, mem1, "New Post", "This is the text", "2006-10-22 13:30:14")**  Once we have our SQL, we can try to run it:  **$result = mysql\_query($tableSQL);**  We can test to see if it is run successfully:  **if ($result) { print "Your Post has been added to the Forum." . "<BR>"; print "<A HREF = forumTest.php>Back to the forum</A>" . "<BR>"; } else { print "Couldn't add Post to the Forum"; }**  If the code executes successfully, a new post will be added to the forum.  And that completes the code for Posting new topics. In fact that completes our walkthrough of the entire forum! There is still work to be done, of course. So there's a final Project for you to try.  **Final Project**  There is no form for a new member to sign up to the forum. Write the code for this, and add links to the signup page in the rest of the forum. Here's a few things you'll need to bear in mind.   * The Table is called **members** * Primary Key is in the format mem1, mem2, mem3, etc * You'll need to build up string for your INSERT INTO SQL * Use PHP to create the Form. Look at the code for **replyForm.php** and**postForm.php** to see how to do this. You'll need to add textboxes for the username and password, and one for the email address. The fields in the members table are:   **memberID  username  password  signupDate  email**   * The form will need to be posted somewhere. You can create a new php page for this. Add the SQL function, and the rest of the code, just like we did above. * You need to bear in mind that the Primary Key in the members table is just like the threadID field, only with mem before the number instead of pos. You'll have to code for this, to make sure you're adding a new row, and not trying to overwrite an existing one * To add the links to the signup page, you'll need to amend the code in a f ew of the pages in the forum, such as **pageReply.php**. A simple hyperlink to your new signup page should do   Play around with the rest of the forum, and see what you can come up with. At the very least, you should be able to improve the look and feel of the forum. If you know any CSS, you can always try to add some to the code.  Speaking of the code, any errors you need to correct in the forum? Most likely. In a larger web application like this, you'll always find a bug or three! And don't forget to keep a copy of the origianl code.    But that completes not only the forum, but the whole beginners PHP course. We hope you enjoyed it, and are motivated to take your new skills on to the next level. Good luck! | | | | | | | | | | | | | |  | | | | | | |