

Introduction

Up until recently, scripting on the internet was something which very few people even attempted, let alone mastered. Recently though, more and more people have been building their own websites and scripting languages have become more important. Because of this, scripting languages are becoming easier to learn and PHP is one of the easiest and most powerful yet.

What Is PHP?

PHP stands for Hypertext Preprocessor and is a server-side language. This means that the script is run on your web server, not on the user's browser, so you do not need to worry about compatibility issues. PHP is relatively new (compared to languages such as Perl (CGI) and Java) but is quickly becoming one of the most popular scripting languages on the internet.

Why PHP?

You may be wondering why you should choose PHP over other languages such as Perl or even why you should learn a scripting language at all. I will deal with learning scripting languages first. Learning a scripting language, or even understanding one, can open up huge new possibilities for your website. Although you can download pre-made scripts from sites like Hotscripts, these will often contain advertising for the author or will not do exactly what you want. With an understanding of a scripting language you can easily edit these scripts to do what you want, or even create your own scripts.

Using scripts on your website allows you to add many new 'interactive' features like feedback forms, guestbooks, message boards, counters and even more advanced features like portal systems, content management, advertising managers etc. With these sort of things on your website you will find that it gives a more professional image. As well as this, anyone wanting to work in the site development industry will find that it is much easier to get a job if they know a scripting language.

What Do I Need?

As mentioned earlier, PHP is a server-side scripting language. This means that, although your users will not need to install new software, your web host will need to have PHP set up on their server. It should be listed as part of your package but if you don't know if it is installed you can find out using the first script in this tutorial. If your server does not support PHP you can ask your web host to install it for you as it is free to download and install. If you need a low cost web host which supports PHP I would recommend HostRocket.

Writing PHP

Writing PHP on your computer is actually very simple. You don't need any special software, except for a text editor (like Notepad in Windows). Run this and you are ready to write your first PHP script.

Declaring PHP

PHP scripts are always enclosed in between two PHP tags. This tells your server to parse the information between them as PHP. The three different forms are as follows:

```
<?
PHP Code In Here
?>
```

```
<?php
PHP Code In Here
php?>
```

```
<script language="php">
PHP Code In Here
</script>
```

All of these work in exactly the same way but in this tutorial I will be using the first option (<? and ?>). There is no particular reason for this, though, and you can use either of the options. You must remember, though, to start and end your code with the same tag (you can't start with <? and end with </script> for example).

Your First Script

The first PHP script you will be writing is very basic. All it will do is print out all the information about PHP on your server. Type the following code into your text editor:

```
<?
phpinfo();
?>
```

As you can see this actually just one line of code. It is a standard PHP function called phpinfo which will tell the server to print out a standard table of information giving you information on the setup of the server.

One other thing you should notice in this example is that the line ends in a semicolon. This is very important. As with many other scripting and programming languages nearly all lines are ended with a semicolon and if you miss it out you will get an error.

Finishing and Testing Your Script

Now you have finished your script save it as phpinfo.php and upload it to your server in the normal way. Now, using your browser, go to the URL of the script. If it has worked (and if PHP is installed on your server) you should get a huge page full of the information about PHP on your server.

If your script doesn't work and a blank page displays, you have either mistyped your code or your server does not support this function (although I have not yet found a server that does not). If, instead of a page being displayed, you are prompted to download the file, PHP is not installed on your server and you should either search for a new web host or ask your current host to install PHP.

It is a good idea to keep this script for future reference.

Part 2

In this part I have introduced you to the basics of writing and running PHP. By this time you should now know if your host supports PHP and should have a basic understanding of how PHP scripts are structured. In part 2 I will show you how to print out information to the browser.

Introduction

In the last part of the tutorial I explained some of the advantages of PHP as a scripting language and showed you how to test your server for PHP. In this part I will show you the basics of showing information in the browser and how you can use variables to hold information.

Printing Text

To output text in your PHP script is actually very simple. As with most other things in PHP, you can do it in a variety of different ways. The main one you will be using, though, is print. Print will allow you to output text, variables or a combination of the two so that they display on the screen.

The print statement is used in the following way:

```
print("Hello world!");
```

I will explain the above line:

print is the command and tells the script what to do. This is followed by the information to be printed, which is contained in the brackets. Because you are outputting text, the text is also enclosed inside quotation marks. Finally, as with nearly every line in a PHP script, it must end in a semicolon. You would, of course, have to enclose this in your standard PHP tags, making the following code:

```
<?
print("Hello world!");
?>
```

Which will display:

Hello world!

on the screen.

Variables

As with other programming languages, PHP allows you to define variables. In PHP there are several variable types, but the most common is called a String. It can hold text and numbers. All strings begin with a \$ sign. To assign some text to a string you would use the following code:

```
$welcome_text = "Hello and welcome to my website.";
```

This is quite a simple line to understand, everything inside the quotation marks will be assigned to the string. You must remember a few rules about strings though:

Strings are case sensitive so \$Welcome_Text is not the same as \$welcome_text

String names can contain letters, numbers and underscores but cannot begin with a number or underscore

When assigning numbers to strings you do not need to include the quotes so:

```
$user_id = 987
```

would be allowed.

Outputting Variables

To display a variable on the screen uses exactly the same code as to display text but in a slightly different form. The following code would display your welcome text:

```
<?
$welcome_text = "Hello and welcome to my website.";
print($welcome_text);
?>
```

As you can see, the only major difference is that you do not need the quotation marks if you are printing a variable.

Formatting Your Text

Unfortunately, the output from your PHP programs is quite boring. Everything is just output in the browser's default font. It is very easy, though, to format your text using HTML. This is because, as PHP is a server side language, the code is executed before the page is sent to the browser. This means that only the resulting information from the script is sent, so in the example above the browser would just be sent the text:

Hello and welcome to my website.

This means, though, that you can include standard HTML markup in your scripts and strings. The only problem with this is that many HTML tags require the " sign. You may notice that this will clash with the quotation marks used to print your text. This means that you must tell the script which quotes should be used (the ones at the beginning and end of the output) and which ones should be ignored (the ones in the HTML code).

For this example I will change the text to the Arial font in red. The normal code for this would be:

```
<font face="Arial" color="#FF0000">
</font>
```

As you can see this code contains 4 quotation marks so would confuse the script. Because of this you must add a backslash before each quotation mark to make the PHP script ignore it. The code would change to:

```
<font face=\"Arial\" color=\"#FF0000\">
</font>
```

You can now include this in your print statement:

```
print("<font face='Arial' color='#FF0000'>Hello and welcome to my website.</font>");
```

which will make the browser display:

Hello and welcome to my website.

because it has only been sent the code:

```
<font face="Arial" color="#FF0000">Hello and welcome to my website.</font>
```

This does make it quite difficult to output HTML code into the browser but later in this tutorial I will show you another way of doing this which can make it a bit easier.

Part 3

In part 3 I will introduce If statements.

Introduction

Over the past two parts I have shown you the basics of text in PHP and how to store it as variables. In this part of the tutorial I will show you how to use IF statements to make decisions in your scripts.

The Basics Of IF

If statements are used to compare two values and carry out different actions based on the results of the test. If statements take the form IF, THEN, ELSE. Basically the IF part checks for a condition. If it is true, the then statement is executed. If not, the else statement is executed.

IF Structure

The structure of an IF statement is as follows:

```
IF (something == something else)
{
    THEN Statement
```

```
} else {  
ELSE Statement  
}
```

Variables

The most common use of an IF statement is to compare a variable to another piece of text, a number, or another variable. For example:

```
if ($username == "webmaster")
```

which would compare the contents of the variable to the text string. The THEN section of code will only be executed if the variable is exactly the same as the contents of the quotation marks so if the variable contained 'Webmaster' or 'WEBMASTER' it will be false.

Constructing The THEN Statment

To add to your script, you can now add a THEN statement:

```
if ($username == "webmaster") {  
echo "Please enter your password below";  
}
```

This will only display this text if the username is webmaster. If not, nothing will be displayed. You can actually leave an IF statement like this, as there is no actual requirement to have an ELSE part. This is especially useful if you are using multiple IF statements.

Constructing The ELSE Statement

Adding The ELSE statement is as easy as the THEN statement. Just add some extra code:

```
if ($username == "webmaster") {  
echo "Please enter your password below";  
} else {  
echo "We are sorry but you are not a recognised user";  
}
```

Of course, you are not limited to just one line of code. You can add any PHP commands in between the curly brackets. You can even include other IF statments (nested statements).

Other Comparisons

There are other ways you can use your IF statement to compare values. Firstly, you can compare two different variables to see if their values match e.g.

```
if ($enteredpass == $password)
```

You can also use the standard comparison symbols to check to see if one variable is greater than or less

than another:

```
if ($age < "13")
```

Or :

```
if ($date > $finished)
```

You can also check for multiple tests in one IF statement. For instance, if you have a form and you want to check if any of the fields were left blank you could use:

```
if ($name == "" || $email == "" || $password == "") {  
echo "Please fill in all the fields";  
}
```

Part 4

In part four I will show you some other ways of using your PHP script to do other types of checks and loops.

Introduction

In the last parts of this tutorial I have showed you how to deal with text and variables in PHP and how you can use IF statements to compare them and to make decisions. In this part I am going to show you how to use another important part of PHP, loops.

The WHILE Loop

The WHILE loop is one of the most useful commands in PHP. It is also quite easy to set up and use. A WHILE loop will, as the name suggests, execute a piece of code until a certain condition is met.

Repeating A Set Number Of Times

If you have a piece of code which you want to repeat several times without retyping it, you can use a while loop. For instance if you wanted to print out the words "Hello World" 5 times you could use the following code:

```
$times = 5;
```

```
$x = 0;
while ($x < $times) {
echo "Hello World";
++$x;
}
```

I will now explain this code. The first two lines are just setting the variables. The \$times variable holds the number of times you want to repeat the code. The \$x variable is the one which will count the number of times the code has been executed. After these is the WHILE line. This tells the computer to repeat the code while \$i is less than \$times (or to repeat it until \$i is equal to \$times). This is followed by the code to be executed which is enclosed in { }.

After the echo line which prints out the text, there is another very important line:

```
++$x;
```

What this does is exactly the same as writing:

```
$x = $x + 1;
```

It adds one to the value of \$x. This code is then repeated (as \$x now equals 1). It continues being repeated until \$x equals 5 (the value of times) when the computer will then move on to the next part of the code.

Using \$x

The variable counting the number of repeats (\$x in the above example) can be used for much more than just counting. For example if you wanted to create a web page with all the numbers from 1 to 1000 on it, you could either type out every single one or you could use the following code:

```
$number = 1000;
$current = 0;
while ($current < $number) {
++$current;
echo "$current<br>";
}
```

There are a few things to notice about this code. Firstly, you will notice that I have placed the ++\$current; before the echo statement. This is because, if I didn't do this it would start printing numbers from 0, which is not what we want. The ++\$current; line can be placed anywhere in your WHILE loop, it does not matter. It can, of course, add, subtract, multiply, divide or do anything else to the number as well.

The other reason for this is that, if the ++\$current; line was after the echo line, the loop would also stop when the number showed 999 because it would check \$current which would equal 1000 (set in the last loop) and would stop, even though 1000 had not yet been printed.

Arrays

Arrays are common to many programming languages. They are special variables which can hold more than

one value, each stored in its own numbered 'space' in the array. Arrays are extremely useful, especially when using WHILE loops.

Setting Up An Array

Setting up an array is slightly different to setting up a normal variable. In this example I will set up an array with 5 names in it:

```
$names[0] = 'John';  
$names[1] = 'Paul';  
$names[2] = 'Steven';  
$names[3] = 'George';  
$names[4] = 'David';
```

As you can see, the parts of an array are all numbered, starting from 0. To add a value to an array you must specify the location in the array by putting a number in [].

Reading From An Array

Reading from an array is just the same as putting information in. All you have to do is to refer to the array and the number of the piece of data in the array. So if I wanted to print out the third name I could use the code:

```
n  
echo "The third name is $names[2]";
```

Which would output:

The third name is Steven

Using Arrays And Loops

One of the best uses of a loop is to output the information in an array. For instance if I wanted to print out the following list of names:

```
Name 1 is John  
Name 2 is Paul  
Name 3 is Steven  
Name 4 is George  
Name 5 is David
```

I could use the following code:

```
$number = 5;  
$x = 0;  
while ($x < $number) {  
  $namenumber = $x + 1;  
  echo "Name $namenumber is $names[$x]<br>";
```

```
++$x;  
}
```

As you can see, I can use the variable \$x from my loop to print out the names in the array. You may have noticed I am also using the variable \$namenumber which is always 1 greater than \$x. This is because the array numbering starts from 0, so to number the names correctly in the output I must add one to the actual value.

Part 5

In the next part I will show you how you can send e-mail using PHP.

Introduction

One of the major uses of a server side scripting language is to provide a way of sending e-mail from the server and, in particular, to take form input and output it to an e-mail address. In this part I will show you how to send e-mail messages using PHP.

The Mail Command

Mail is extremely easy to send from PHP, unlike using scripting languages which require special setup (like CGI). There is actually just one command, mail() for sending mail. It is used as follows:

```
mail($to,$subject,$body,$headers);
```

In this example I have used variables as they have descriptive names but you could also just place text in the mail command. Firstly, \$to. This variable (or section of the command) contains the e-mail address to which the mail will be sent. \$subject is the section for the subject of the e-mail and \$body is the actual text of the e-mail.

The section \$headers is used for any additional e-mail headers you may want to add. The most common use of this is for the From field of an e-mail but you can also include other headers like cc and bcc.

Sending An E-mail

Before sending your mail, if you are using variables, you must, of course, set up the variable content beforehand. Here is some simple code for sending a message:

```
$to = "php@gowansnet.com";  
$subject = "PHP Is Great";  
$body = "PHP is one of the best scripting languages around";  
$headers = "From: webmaster@gowansnet.com\n";  
mail($to,$subject,$body,$headers);  
echo "Mail sent to $to";
```

This code will actually do two things. Firstly it will send a message to php@gowansnet.com with the subject 'PHP Is Great' and the text:

PHP is one of the best scripting languages around

and the e-mail will be from webmaster@gowansnet.com. It will also output the text:

Mail sent to php@gowansnet.com

to the browser.

Formatting E-mail

Something you may have noticed from the example is that the From line ended with \n. This is actually a very important character when sending e-mail. It is the new line character and tells PHP to take a new line in an e-mail. It is very important that this is put in after each header you add so that your e-mail will follow the international standards and will be delivered.

The \n code can also be used in the body section of the e-mail to put line breaks in but should not be used in the subject or the To field.

Mail Without Variables

The e-mail above could have been sent using different variable names (it is the position of the variables in relation to the commas, not the name of them which decides on their use). It could also have been done on one line using text like this:

```
mail("php@gowansnet.com","PHP Is Great","PHP is one of the best scripting languages around","From: webmaster@gowansnet.com\n");
```

But that would make your code slightly harder to read.

Error Control

As anyone who has been scripting for a while will know, it is extremely easy to make mistakes in your code and it is also very easy to input an invalid e-mail address (especially if you are using your script for form to mail). Because of this, you can add in a small piece of code which will check if the e-mail is sent:

```
if(mail($to,$subject,$body,$headers)) {  
echo "An e-mail was sent to $to with the subject: $subject";  
} else {  
echo "There was a problem sending the mail. Check your code and make sure that the e-mail address $to is valid";  
}
```

This code is quite self explanatory. If the mail is sent successfully it will output a message to the browser telling the user, if not, it will display an error message with some suggestions for correcting the problem.

Part 6

In part 6 I will continue covering mail by showing you how to make a simple form to mail program in PHP.

Introduction

In the last part, I showed you how to use PHP to send e-mail messages using a script. In this part I will continue this and also show you how to use PHP and forms together to make your PHP scripts useful.

Setting Up Your Form

Setting up a form for use with a PHP script is exactly the same as normal in HTML. As this is a PHP tutorial I will not go into depth in how to write your form but I will show you three of the main pieces of code you must know:

```
<input type="text" name="thebox" value="Your Name">
```

Will display a text input box with Your Name written in it as default. The value section of this code is optional. The information defined by name will be the name of this text box and should be unique.

```
<textarea name="message">
Please write your message here.
</textarea>
```

Will display a large scrolling text box with the text 'Please write your message here.' as default. Again, the name is defined and should be unique.

```
<input type="submit" value="Submit">
```

This will create a submit button for your form. You can change what it says on the button by changing the button's value.

All the elements for your form must be enclosed in the <form> tags. They are used as follows:

```
<form action="process.php" method="post">
Form elements and formatting etc.
</form>
```

The form's action tells it what script to send its data to (in this case its process.php). This can also be a full URL (e.g. <http://www.mysite.com/scripts/private/processors/process.php>). The method tells the form how to submit its data. POST will send the data in a data stream to the script when it is requested. GET is the other option. GET will send the form data in the form of the url so it would appear after a question mark e.g. <http://www.mysite.com/process.php?name=david>

It really makes no difference which system you use but it is normally better to use POST if you are using

passwords or sensitive information as they should not be shown in the browser's address bar.

Getting The Form Information

The next step is to get the data the form has submitted into your script so that you can do something with it. This is. There are basically two different methods of getting the data into PHP, which depend on how they were submitted. There are two submission methods, GET and POST, which can both be used by forms. The difference between the two is that using GET, the variables and data will be shown in the page address, but using POST it is invisible. The benefit of GET, though is that you can submit information to the script without a form, by simply editing the URL.

This works the same as submitting a form using GET. The advantage of this is that you can create links to your scripts which do different things depending on the link clicked. For example you could create a script which will show different pages depending on the link clicked:

[yourpage.php?user=david](#)

could show David's page and:

[yourpage.php?user=tom](#)

could show Tom's page, using the same script.

It is also possible to pass more than one piece of information to the script using this system by separating them with the & symbol:

[yourpage.php?user=david&referrer=gowansnet&area=6](#)

These could all be accessed separately using the GET variables user, referrer and area.

To get a variable which has been sent to a script using the POST method you use the following code:

```
$variablename=$_POST['variable'];
```

which basically takes the variable from the POST (the name of a form field) and assigns it to the variable \$variablename.

Similarly, if you are using the GET method you should use the form:

```
$variablename=$_GET['variable'];
```

This should be done for each variable you wish to use from your form (or URL).

Creating The Form To Mail Script

To finish off this section, I will show you how to use what you have learnt in this part and the last to create a system which will e-mail a user's comments to you.

Firstly, create this form for your HTML page:

```
<form action="mail.php" method="post">
Your Name: <input type="text" name="name"><br>
E-mail: <input type="text" name = "email"><br><br>
```

```

Comments<br>
<textarea name="comments"></textarea><br><br>
<input type="submit" value="Submit">
</form>

```

This will make a simple form where the user can enter their e-mail address, their name and their comments. You can, of course, add extra parts to this form but remember to update the script too. Now create the PHP script:

```

<?
function checkOK($field)
{
if (ereg("r",$field) || ereg("n",$field)){
die("Invalid Input!");
}
}

$name=$_POST['name'];
checkOK($name);
$email=$_POST['email'];
checkOK($email);
$comments=$_POST['comments'];
checkOK($comments);
$to="php@gowansnet.com";
$message="$name just filled in your comments form. They said:\n$comments\n\nTheir e-mail address
was: $email";
if(mail($to,"Comments From Your Site",$message,"From: $email\n")) {
echo "Thanks for your comments.";
} else {
echo "There was a problem sending the mail. Please check that you filled in the form correctly.";
}
?>

```

Remember to replace php@gowansnet.com with your own e-mail address. This script should be saved as mail.php and both should be uploaded. Now, all you need to do is to fill in your comments form.

The first part of that script may look a bit strange:

```

function checkOK($field)
{
if (ereg("r",$field) || ereg("n",$field)){
die("Invalid Input!");
}
}

```

You don't really need to worry about what this is doing, but basically, it stops spammers from using your form to send thier spam messages by checking special characters are not present in the input which can be used to trick the computer into sending messages to other addresses. It is a fuction which checks for these characters, and if they are found, stops running the script.

The lines:

```
checkOK($name);
```

etc. run this check on each input to ensure it is valid.

Part 7

As you can see, using forms with PHP can be very effective. In the next part I will show you some of the final things you should know about PHP.

Introduction

In the past 6 parts of this tutorial I have shown you the basics of writing PHP. In this final part I will show you a few small things which don't really warrant a section of their own.

Comments

As with any programming language, it is quite important to comment in your script. If you are working on a script with someone else you must let them know what you code does and if you are distributing your script you will need to show people how to edit it. Even if you are the only one who will use your script it is useful to comment so that you can edit it at a later date.

In PHP there are two ways you can comment. One way is used for single line comments and the other is used mainly for comments that go over one line. A single line comment is written as follows:

```
// Your comment can go in here
```

Everything after the // will be ignored when the script is executed. You can even place these on the end of another line e.g.

```
print "Hello $name"; // Welcome to the user
```

Another way of commenting is by using multi-line comments:

```
/* The following piece of code will take the input  
the user gave and will check that it is valid before  
adding it to the database */
```

Anything between the /* and the */ will be ignored. It is important that you always close this type of comment as not doing so could make your script not work.

Print, Echo and HTML

As you may have noticed during this tutorial I have actually used 4 different ways of outputting information to the browser:

```
echo("Text here");
echo "Text here";
print("Text here");
print "Text here";
```

To clarify, all of these do the same thing and you can use any or all of them in a script. There is no reason to even use the same type all through a script. The only problem you may find is that, as I explained in part 2, all the " in the HTML code must be replaced with \" which, if you have a lot of code, could take a very long time. This brings me to a very useful part of PHP. If, for example, you created the header of a page dynamically in PHP, then had the static page and finally a dynamic footer you can do the following:

```
<?
Top PHP code in here
?>
HTML Code
<?
Bottom PHP code in here
?>
```

This gets even better as the PHP code will just continue from where it was left off so you could do the following:

```
<?
IF Statement {
?>
HTML For IF Being Correct
<?
} else {
?>
HTML For IF Being Wrong
<?
}
?>
```

You must always remember to close IF statements and loops, though, as it is very easy to forget.

One Line Prints

Being able to place HTML code into your PHP is very useful, but what happens if you want to put the value of a variable into the code. Unlike when using an echo or print statement, you can't just put in the variable name as this section is not actually part of the PHP code. Instead you must just put in a little PHP.

For example if you wanted to print someone's name from a script with HTML formatting you would do the following:

```
<font face="Arial" size="7" color="red"><b><? echo($variablename); ?></b></font>
```


In the above code you have just added in the following PHP:

```
<? echo($variablename); ?>
```

Which is exactly the same as the following PHP code:

```
<?  
echo($variablename);  
?>
```

But all put onto one line.

Conclusion

This tutorial has given you some of the basics of PHP and should allow you to do most things you will want to. For a much more in depth look you should visit PHP.net, the official homepage of PHP. One major omission of this tutorial, you may have noticed, is using PHP with a database. As this is one of the major reasons that people use PHP and because there are many options I will put this in a separate PHP/MySQL tutorial.