PHP IS A powerful scripting language that fits gracefully into HTML and puts the tools for creating dynamic websites in the hands of the people — even people like me who were too lazy to learn Perl scripting and other complicated backend hoodoo. This tutorial is for the person who understands HTML but doesn't know much about PHP. One of PHP's greatest attributes is that it's a freely distributed open-source language, so there's all kinds of excellent reference material about it out there, which means that once you understand the basics, it's easy to find the materials that you need to push your skills.

INTRODUCTION

What Is PHP?

So, what is this whole PHP business all about?

PHP is a program that gets installed on top of your web server software. It works with versions of Apache, Microsoft IIS and other server software packages.
You use PHP by inserting PHP code inside the HTML that makes up your website. When a client (anybody on the web) visits a web page that contains this code, your server executes it. That's why you need to install your own server in order to test PHP locally — the server is the brain here, not your browser. Users don't need any special plug-ins or anything to see your PHP in action — it gets to the end user as regular old-fashioned HTML.

PHP is a scripting language, like HTML. That means that code does not need to be compiled before it gets used — it gets processed on the fly as necessary.

Before we dig in, you should know about a site called PHP.net. PHP is an open-source language, and PHP.net is its control center, with extensive reference material about the language and tips sent in by users across the globe. PHP.net has exceptional, deep information about the language, but it can be a little cryptic for the newcomer. We'll look more closely at how to use PHP.net at the end of this tutorial.
So, what kinds of things can PHP do? Welllll … it can:

- take info from web-based forms and use it in a million ways (store it in a database, create conditional pages depending on what the forms said, set cookies for later, send e-mail, write your mom on her birthday);
- authenticate and track users;
- run threaded discussions on your site;

- serve different pages to people using different browsers or devices;
- publish an entire website using just a single layout template (server-side includes-style);
- serve XML pages.

But before we can get to the specific uses of PHP, we need to start with a quick preview of the building blocks of PHP, beginning with a sample script. This example script is titled "chickenman.php." When called by a web browser, it would simply read, "I am the CHICKEN MAN!"

```
<?php print ("I am the CHICKEN MAN");  ?>
```

The ?php and ?¢ tags start and end a PHP script, and your meat goes in the middle. Got that? Good! Now let's walk through the basic rules you need to know to before you can write your first PHP script.

WHAT YOU'LL NEED

Before we begin, you will need to install a server on your own machine in order to test your PHP scripts locally. You can install WampServer for Windows machines from http://www.wampserver.com/en/ In order to have a Localhost machine. If you're using a Mac you can get MAMP from http://www.mamp.info.
If you have space on a web server which supports PHP, you can also test your PHP there, but this is kind of a pain because it means you'll need to FTP your files or telnet in every time you want to change something.

STEPS

The Basics

The code itself fits right inside a page's HTML, and like HTML it is made up of plain ol' text. So a page that displays the words "I am the CHICKEN MAN!" message would sit inside an HTML page named something.php, like this:

```
<html> <head> <title> Chicken Man Example </title> </head> <body> <font
color="red">My PHP code makes this page say:</font> <p> <?php print ("I am
the CHICKEN MAN"); ?> </p> </body> </html>
```

See how that works? The HTML is rendered as regular HTML, but everything inside the ?php and ?¢ tags gets processed as PHP.

Basic Syntax
It’s time to write your own first PHP script. The basic rules of PHP are as follows:

Naming Files

In order to get a PHP script working, the file it’s in or the file that it calls to do the heavy lifting must end in .php (earlier versions used the file extensions .php3 and .phtml). Like HTML, your files are saved as plain text.

Comments

It’s important to get in the habit of leaving notes about your code with the comment tags so that months down the road you can make sense of what you were trying to make your script do. The way you set comments apart from your code (that you don’t want displayed or executed) is with either “//” at the beginning of each line, or surrounded by “/*” and “*/” if you want to comment out several lines:

```
<?php // This will be ignored. Note to self: // Pick up ice cream, cake, and
balloons. print ("I am the CHICKEN MAN"); /* This, too, will be ignored. Hey,
and don't forget the spanking machine! */ ?>
```

Code Syntax
Start of Code

Every piece of PHP code begins with “<?php” (or the abbreviated “<?” if your server is configured to handle that).

End of Code

The way to signify that the PHP code is finished is by adding “?>” at the end.

Every Chunk

With a few exceptions, each separate instruction that you write will end with a semicolon.

Parentheses

The typical function looks like this …

```
print ( );
```

… where "print" is the function and the stuff that the function works on sits inside the parentheses, with a semicolon to finish it off. (Just to confuse you, "print" is the exception that also works without parentheses.) By the way, echo () is the same as print ().

Much like HTML, the actual formatting of your PHP code (where you put spaces, line breaks, etc.) will not affect the outcome except those parts of the code that tell a web browser how to display your page. So this piece of code …

```
<?php print ("I am the CHICKEN MAN");  ?> ... is effectively identical to:
<?php print ("I am the CHICKEN MAN"); ?>
```

Like more complicated HTML, it behooves you to use white space and tabs in your code to make the code more understandable.

Ready to write your first script? Let's go.

Your First Script

OK, so write your first script already! Copy the following script, but put whatever you want inside the quotation marks. "Print" here means print to the screen of a web browser when you open the file:

```
<html> <body> <?php print ("I am the CHICKEN MAN");  ?> </body> </html>
```

Save the file with any name that has no spaces and ends in .php, and if you've installed a server on your own machine, you need to save the script somewhere inside the server's root folder (on Windows this is typically in the "wwwroot" directory inside the "inetpub" directory on your C: drive).

The next step is to open the file in your browser. Since you need the server to run your PHP code, you have to open the file through a URL that finds the correct file through your web server. On Windows, your computer name is your root URL. My computer name is "rocketboy," so to see the contents of my root directory, I type "http://rocketboy" into the Web browser and voila! I see the contents of my root folder. To open the file "chickenman.php" in a directory called "tests" inside the root directory, I'd type "http://rocketboy/tests/chickenman.php" and see my example.

If you're testing on a PHP-able web server, FTP your files anywhere on your server and they should work when you open them through the URL.

Go on now and get your first script working. Then come back and we'll have some fun. Together.

Error Messages

Fun, eh? Fun if it worked. If not — if you had an error in your script — you probably got an error message that looked something like this:

```
Parse error: parse error in C:Inetpubwwwrootwebmonkey_articletest9.php on
line 12
```

Error messages can be very useful and you're bound to run into lots of them. You'll get a message like this for every line in your script that has an error. For our purposes, all we really need to know is that there is something wrong with our code in line 12 of the document "test9.php," so let's look at that line and see if we can figure it out (good text editors like BBEdit have a function that lets you jump to any particular line). I always start by looking to see if my basic syntax is correct: did I leave out the closing tag, a line's semicolon, quotation marks?

A Few More Statements

Let's continue by adding to your test code from the last page to show a couple useful tools.

In the same code that you wrote before, drop in a couple more statements. As you see, you can gang up more than one PHP function inside the same opening and closing tags. My comments in the code explain what each part does:

```
<html> <body> This text right here (or any HTML I want to write) will show up
just before the PHP code stuff. <p> <?php // first, this $PHP_SELF thang is
// an environment variable that'll show the // path from your root folder to
this // document itself, like /webmonkey_article/test3.php. // I put this in
just for fun. // NOTE: This may only work if your server is Apache. print
"$PHP_SELF"; // next we have to "print" any // HTML code we want the browser
// to follow to determine // the layout of the results page. // In this case,
we're adding a <p> tag // the <p> tags could have been put // inside the same
print statement as the // "I am the CHICKEN MAN" text. // between the
$PHP_SELF text and the // next bunch of stuff. print ("<p>"); print ("I am
the CHICKEN MAN"); print ("<p>"); /* This next "phpinfo" business outputs a
long page that tells you exactly how your version of PHP is configured. This
can be useful when troubleshooting problems down the road */ phpinfo();  ?>
</p> </body> </html>
```

NOTE: Phpinfo will output a long page of info about your version of PHP. You don't need to understand what it all means, I just wanted to show you that it's there if you ever need it.

Very Able Variables

So far, all we've done is have a PHP script print some text. Big whoop. Let's get down and dirty now with variables. A variable is a container for holding one or more values. It is the means by which PHP stores information and passes it along between documents and functions and such. You may remember variables from algebra — in the equation "x + 2 = 8", x is a variable with the value 6.

The reason why variables are so important to PHP is that the very notion of having dynamic web pages — pages which respond somehow to user input — relies on data being passed around between pages (or parts of a page). Variables are the main mechanism for transferring data like this.

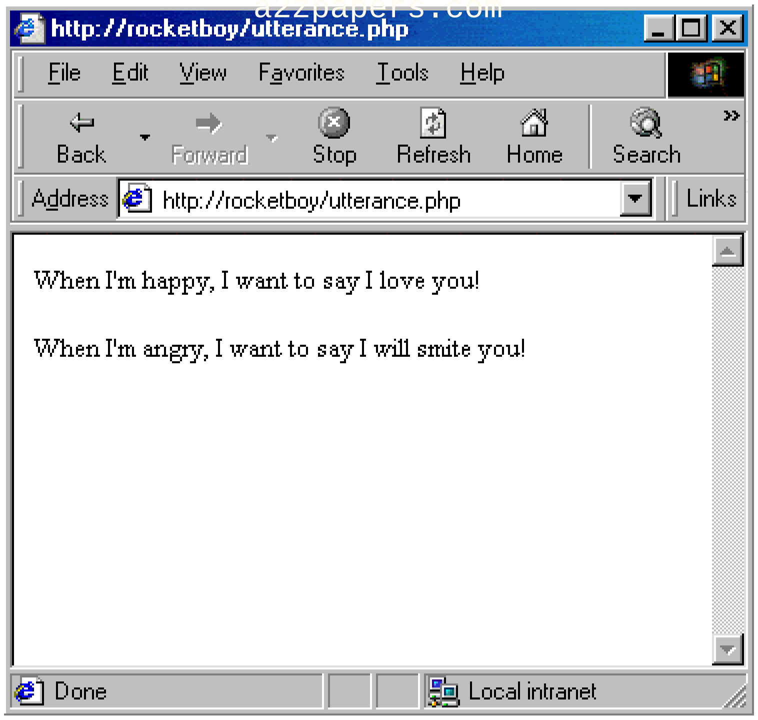
I think the easiest way to explain how variables work in PHP is to show them in action. There are three basic things you can do with variables:

1. Set them (give them one or more values);
2. Re-set them if they were set before;
3. Access them (read the value of a variable and then do something useful with it).

Variables in PHP start with a dollar sign ("$"). Below I am setting a variable, using it, then setting and using it again. The value that a variable holds can be changed any time at all.

```
1. <?php 2. $utterance = "I love you!"; 3. print ("When I'm happy, I want to
say $utterance"); 4. print ("<p>"); 5. $utterance = "I will smite you!"; 6.
print ("When I'm angry, I want to say $utterance"); 7. ?>
```

Here's what that page will look like:


http://rocketboy/utterance.php
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://rocketboy/utterance.php Links
Done Local intranet

```
When I'm happy, I want to say I love you!

When I'm angry, I want to say I will smite you!
```

In line two I have created a variable that I decided to name "utterance." All variables start with "$", so my variable is written "$utterance" in the code. Here's how that last code snippet breaks down line by line. Please note: the webserver does the PHP interpreting before sending the browser finished HTML code.

- Line 1 tells the webserver: “Start PHP code here”.
- Line 2 creates the variable $utterance and also sets it, giving it the initial value of “I love you!”.
- Line 3 prints a phrase that draws on the variable $utterance.
- Line 4 creates a

  tag in HTML to put vertical space between the two utterances.
- Line 5 RE-SETS the variable $utterance and gives it the value “I will smite you!”.
- Line 6 prints a new phrase that draws on the new meaning of the variable $utterance.
- Line 7 tells Mr. Webserver: PHP code ends here.

See how the variable $utterance is used as a sort of container that can hold different values? We just set and then called variables inside the same script, but the power of PHP is that you can set a variable in one place — say from a form that a user fills out — and then use that variable later.

The syntax of setting a variable is to:

- define it with the = sign ($utterance = “I love you!”;);
- use quotation marks if you’re defining it with a string of letters (“I love you!”; numbers don’t require quotes);
- end each instruction with a semicolon.

Then you call it by refering to the variable name ($utterance in lines 3 and 6 — notice no quotation marks there).

Naming Variables
You can name a variable anything you want so long as it follows these rules:

- it starts with a letter;
- it is made up of letters, numbers, and the underscore character (that’s the _ character, as in “$baby_names”);
- it isn’t used elsewhere (like “print”).

Warning: Variable names are case-sensitive, so $baby_names and $Baby_names are not the same. You also should try to make your names have some meaning so that you can still make sense of your code next year.

In the examples so far, we have set variables as chunks of text, which are known as "strings." Variables can also hold the values of numbers and some other things as well (objects, arrays, booleans).

Final note: One thing that can be a little confusing when starting to use PHP is the use of quotation marks inside PHP functions. Use single or double quotes to set off strings (that is, chunks of text), as in:

```
print ("I am the CHICKEN MAN");
```

This will print the text I am the CHICKEN MAN. If you want to display quotation marks as characters in the output of your PHP script, you must escape them with the "" character, which tells PHP not to use the next character as part of the code. So to output the text "I am the CHICKEN MAN" (with quotation marks showing in the result) the code would look like:

```
print (" "I am the CHICKEN MAN"" );
```