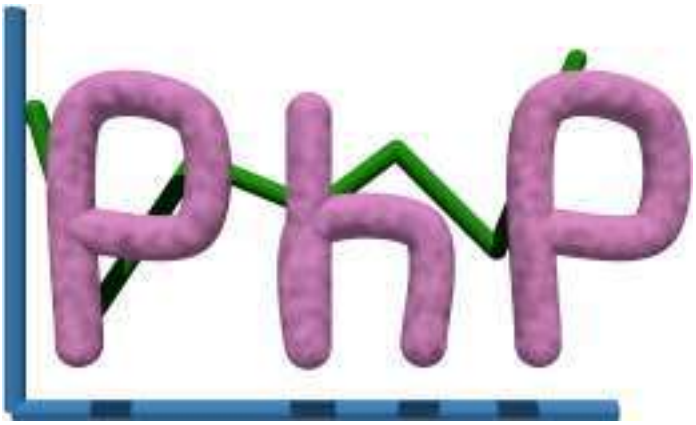
PHP for Dynamic Web Pages



by Jerry Stratton
Friday, September 12, 2008

http://www.hoboes.com/NetLife/PHP/
http://www.hoboes.com/NetLife/PHP/PHP.pdf

Contents

WHY USE PHP? 1

USING PHP ON APACHE AND UNIX 2

The php Extension 2

PHP CODE 3

Functions 3
Containers 3
PHP With Forms 4
POST and GET 5
Conditional HTML 6
PHP With E-Mail 8
A Note About Email 9
PHP With Files 9
Error Reporting 11
Write Access 11
Arrays 12
Accessing Array Data 13
Don't Trust Anyone Outside 14
Sorting Arrays 15
Visitor Sessions 15
Requiring Cookies 17
Sessions and Security 17
The Program So Far 17
Creating Images On the Fly 18
Creating Your Own Functions 20
Optional Arguments 22

WORKING WITH MYSQL 24

Create the table 24
Create the Database 24
Create the Username and Password 24
Create the Table 24
Store the data 25
Display the data 26

CURLY BRACKETS 28

GNU FREE DOCUMENTATION LICENSE 29

0. Preamble 29
1. Applicability and Definitions 29
2. Verbatim Copying 30
3. Copying in Quantity 30
4. Modifications 31
5. Combining Documents 32
6. Collections of Documents 32
7. Aggregation With Independent Works 32
8. Translation 32
9. Termination 33
10. Future Revisions of this License 33

PHP: HYPERTEXT PROCESSOR 34

Why Use PHP?

If you need to embed dynamic text into static text, you'll find PHP extremely useful. It was designed for this, and it excels at it. PHP is also very useful for integrating web pages with databases.

The PHP scripting language resembles JavaScript, Java, and Perl, These languages all share a common ancestor, the C programming language.

PHP is most different from JavaScript and Java. PHP is a *server-side* scripting language. All of the "work" is done on the server. JavaScript (and Java) generally run on the client. They have little access to the information that the server has, and mediated access to information on the client. They can do lots of things on the client that PHP cannot. PHP has full access to the information that the server has, and very little access to information that the client has. In fact, it only has information that the client tells the server and that the server passes on to PHP. Because it is on the server, however, PHP cannot be modified by the client. While you cannot necessarily trust the information that the client gives to PHP, you can trust that your PHP is doing what you told it to do. Because PHP is on the server end, your PHP scripts can affect your server—such as by keeping an activity log or updating a database.

PHP and Perl often work side-by-side. These are both server-side. Where PHP excels at embedding dynamic content, Perl excels at modifying (or "filtering") streams of text. PHP excels at putting things into documents, and Perl excels at finding things in documents. After you have learned PHP, you may well find Perl useful for many tasks, especially for command-line tasks. PHP has an advantage over Perl on most web sites because PHP is usually loaded as part of the web server. When scripting languages "run", the system has to first load the "interpreter" and then "compile" the language into code that the machine can understand. When you tell PHP to echo the current time to the web page, the computer needs to have your command translated into numbers that it can understand. Because the PHP interpreter is already loaded as part of the web server's software, it is always running. This cuts out half of that process. The interpreter is already loaded, and it can go directly to compiling the language into code. When web servers see a request to run a Perl script, they usually have to first load the Perl interpreter. This happens very quickly, but when there are thousands or tens of thousands of requests coming every second, every "very quickly" can add up.

C programs are "pre-compiled". They cut out both steps in that process: no interpreter is needed because the program is already compiled into code the machine understands. Because of this, however, C programs must be compiled every time you switch to a new machine. If you move to a different host, you will usually have to recompile your C programs. Sometimes you'll even have to recompile your C programs when your ISP upgrades their server's system software. And many ISPs do not provide you with a C compiler. You'll find that PHP is more "portable" than C in this respect: if it works on one server, it will usually work on any other server that has it. Most ISPs that provide server-side scripting provide PHP.

Using PHP on Apache and Unix

You can get more information about PHP, as well as the full PHP manual, at the PHP web site, http://www.php.net/. You can also subscribe to the PHP mailing list there. The PHP on-line manual is extremely useful: not only does it let you quickly look up any part of PHP, but it includes notes from people who use PHP about problems you might run into and how to fix them.

If you want more information about PHP in printed form, there is "Programming PHP" by Rasmus Lerdorf and Kevin Tatroe. Rasmus Lerdorf is one of the authors of the PHP language.

If you are interested in using PHP with the MySQL database there is also "Web Database Applications with PHP & MySQL" by Hugh E. Williams and David Lane, or "MySQL" by Paul DuBois. I've found the latter to be an extremely useful MySQL reference.

The php Extension

If you've used server-side includes, you know that you usually have to rename your web pages from "something.html" to "something.shtml" in order to get the server to "do" the includes. The extension "shtml" lets the server know that it has more work to do on the web page before the reader gets to see it. Similarly, your "PHP" web pages usually need to end in the extension "php". This lets the server know that it needs to hand this page off to the "php" module before letting the reader see the web page.

Some servers will be set up to require an extension of ".php3" instead of ".php". The choice of extension is completely up to whoever sets up your server. While it will usually be "php", it can be anything. Contact your system administrators to be sure.

PHP Code

PHP code starts out looking like HTML code and ends up looking nothing like it. If you've looked at HTML code, you've seen things that look like "" or "<h2>". You will put your PHP code between "<?" and "?>". For example, the following web page will display the current time:

```
<html>
    <head>
        <title>My PHP Page</title>
    </head>
    <body>
        <h1>PHP Test Page</h1>
        <p>The current time is <?echo date('h:i A')?></p>
    </body>
</html>
```

Except for the php part, this looks like a normal web page. The only PHP part is the "<?echo date('h:i A')?>". If you save this file as "test.php" on your web site, and then view it, you will see the current time every time you reload your page.

Functions

PHP uses *functions* to get things done. The functions we just used are the "echo" function and the "date" function. (In Unix, there is no difference between dates and times. They're all stored as one number.) The 'echo' function is simple. Whatever you give it, it 'echoes' to the web page. In this case, we gave it the 'date' function. The 'date' function (http://www.php.net/manual/function.date.php) returns the current date and time in whatever format you want. The format code we used was "h:i A". This means we want the hour, a colon, the minute, a space, and either AM or PM. If it is currently March 9, 10:11 AM and 14 seconds, this will give us "10:11 AM", which the "echo" function inserts as part of the web page.

Containers

Rather than immediately echoing the results of functions, you can store them in containers for later use. (These containers are often called "variables" by programmers.) If, for example, you were going to mention the time more than once on your web page, you might want to store the time in a container and simply echo that container everywhere on your page. Change the body of your web page to:

```
<h1>PHP Test Page</h1>
<?
    $now = date('h:i A');
?>
<p><em>Right now</em> is <?echo $now?>. <?echo $now?> is a very important time,
because <?echo $now?> is the exact time you visited our wonderful web page.</p>
```

The two main advantages of using containers is that it reduces errors by making it so that you only have to type the date function once, and it gives you the option of changing your mind about the format later. Suppose later on you want to change the time format to 24-hour time. You can change the date format to 'H:i' on the one line, $now = date('l, F jS, Y h:i A');, and it will change to showing the current weekday, the date, and the time in all three places.

We've put the time in a *container* called '$now'. All containers in PHP begin with the dollar sign.

PHP With Forms

PHP is designed to be used with HTML forms. Add the following to the body of your test file:

```
<?
    $color = $_REQUEST["color"];
    IF ($color):
?>
    <p style="color: <?echo $color?>">
        You said your favorite color was <?echo $color?>.
    </p>
<?ENDIF?>

<form method="post" action="test.php">
    <p>
        What is your favorite color?
        <input type="text" name="color" value="<?echo $color?>" />
    </p>
    <input type="submit" />
</form>
```

This is a one-field form. It just asks for the reader's favorite color. The form's "action" is the file itself: any php file can also be a form interpreter. The only field we have is the field 'color', so PHP automatically creates a container called "$color" which contains what the reader typed in that field.

The line "$color = $_REQUEST["color"];" tells PHP to take the "request" called "color" and place that in the container called "$color". PHP puts form data into a list called "$_REQUEST". You can ask for each form item by name in that list.

The line "IF ($color):" tells PHP that we only want to do the next few lines (until the "<?ENDIF ?>") *if* the variable "$color" exists and has something in it. If this is the first time the reader viewed the page, or the reader pressed the submit button without typing a favorite color, "$color" will be empty, and those lines will be skipped.

We make use of the "echo" function to set the color of the "You said…" line, and to pre-fill the field if they've already filled out the form once.

Notice that we moved in and out of HTML in this example. We start with PHP, switch—while still in the "if" area—to HTML, and then switch back to PHP.

Sometimes you'll want to know whether the field exists rather than, or in addition to, whether it actually has anything in it. You can use a function called "isset" to determine this. Rewrite your web page so that it reads:

```
<?
    $color = $_REQUEST["color"];
    IF (isset($color)):
        IF ($color == ""):
            echo "<p>You need to enter a color!</p>\n";
        ELSE:
?>
    <p style="color: <?echo $color?>">
        You said your favorite color was <?echo $color?>.
    </p>
<?
        ENDIF;
    ELSE:
        echo "<p>Welcome to our color extravaganza!</p>\n";
    ENDIF;
?>

<form method="post" action="test.php">
    <p>
        What is your favorite color?
        <input type="text" name="color" value="<?echo $color?>" />
    </p>
    <input type="submit" />
</form>
```

You'll notice a couple of changes here. First, we have an IF inside of our IF block. This is perfectly reasonable, and you will often do this. First, we see if the container "$color" is "set", that is, has it been used at all. If it has, we go ahead and decide whether it has anything in it. In this case, we specifically check to see if it contains "", that is, nothing.

Here, we used two equal signs. This is the source of one of the most common mistakes in programming. When we set a container to another value, we use a single "=". When we check to see what a container contains, we use a double "==". If you use one in place of the other, you will have major problems.

If our $color container contains nothing, we tell them they need to enter a color. Otherwise, we display their color.

Go ahead and try this script out. When you first visit the page, it should welcome you. If you try to submit the form with no color, it should tell you that you need to enter a color.

POST and GET

There are a number of "methods" that you can use to send your form data to the server for PHP to parse. Two of the most common are "POST" and "GET". These each have their own place. The GET method is very useful if you want the viewers to be able to "come back" to the results page. For example, if you are providing a list of rooms in a building, you might use "GET" to allow them to bookmark a specific building. Or if you are providing a form that lets them search a list of classes by topic, you might use GET so that they can bookmark the topic and come back to it later to see if there are any new classes in that topic.

The GET method also allows them to copy the URL out of their web browser and send it by e-mail. So they could, for example, look up classes on a certain topic and then e-mail a link to those classes to a friend. Search engines often use the GET method. This allows viewers to bookmark certain searches, and it allows them to send the search results to friends or colleagues.

The POST method is not bookmarkable. You should use POST if you do *not* want the user to be able to "come back" to this page. For example, if they are purchasing something you don't want them to submit the purchase twice. If they are deleting something from your database, coming back a second time will probably just result in an error; if they are inserting something into your database, they may end up inserting it twice if you aren't careful with your PHP code.

POST information is also somewhat more secure. GET information is part of the URL. This means that it is also stored in the web server's logs. Anyone who can see those logs can see the form information. Even if the server itself is a secure server, GET information is still posted to the logs. If your form requests secure or private information, you should use POST to submit it.

If they are going to be submitting a *lot* of data, you will need to use POST. Web browsers and web servers can "truncate" GET submissions. The limit on the size of a GET submission is highly variable but the general recommendation is that if the form data is likely to approach 1,024 bytes, go with the POST method. This is probably part of the reason that search engines, which want to be bookmarkable, will abbreviate their form fields to two-letter or one-letter fieldnames.

Conditional HTML

We've done a little bit of conditional HTML—HTML that only gets sent to the browser depending on other conditions, such as whether the browser submitted the form. Let's take a closer look at this.

Try typing, as your favorite color, the color of your browser's background. On my browser, this is white. Our PHP code will dutifully display the text in white, causing it to be completely invisible against the browser's white background. We don't have to stand for this behavior. Let's make sure that the example color is always visible.

Change the lines that say:

```
        ELSE:
?>
    <p style="color: <?echo $color?>">
```

to

```
        ELSE:
            IF ($color == "white"):
                $bgcolor = "grey";
            ELSE:
                $bgcolor = "white";
            ENDIF;
```

```
?>
   <p style="color: <?echo $color?>; background-color: <?echo $bgcolor?>;">
```

Now, if you type in “white”, the background color will be grey. If you type in anything else, the background color will be white.

Works great, right?

Go ahead and type “White” or “WHITE”. The text is invisible again. PHP, like most programming languages, is case sensitive. But cascading style sheets (and, currently, HTML) is not. This means that when PHP checks to see if “White” is the same as “white”, it says that it is not. But when we set the color to “White” and the background color to “white”, CSS gives the text and the background of our paragraph the exact same color.

We can alleviate this problem by converting the color they choose to all lower case. This way, when we check their color against our color, case will no longer matter—theirs will be all lower case as well. Change the “IF ($color == “white”):“, line to the following:

```
$color = strtolower($color);
IF ($color == "white"):
```

Try it again, and now, a favorite color of “white”, “White”, or even “wHiTE” all result in visible text.

Try typing white with a space after it. Again with the invisible text! PHP is checking exactly, but CSS (like HTML) ignores what is called “white space”—spaces, for example. Well, we can fix this as well. Add one more line:

```
$color = strtolower($color);
$color = trim($color);
IF ($color == "white"):
```

That works, but this is getting tedious. Where does it end? The real problem here is that we are beginning to assume things about the browser’s state and the user’s actions that we cannot assume. We thought it would be nice to show an example of the color the viewer chose. Then we thought it might be a good idea to make sure that they can always see the example color. But on the web, we never have control over the user’s browser, and if we know what we are doing we don’t want it.

There are other ways around this problem. We could, for example, keep a list of valid colors. If the color they choose is not in that list, do not show them an example color. This ensures that when they find a way of representing white that we haven’t anticipated, we do not end up showing them invisible text. In fact, it fixes a whole bunch of potential problems called “cross-scripting” as well. What we’re doing is inserting into “our” HTML some text that the viewer has “typed”. We are assuming that humans are typing this and that they are typing colors. In fact, however, we cannot assume *anything* about the stuff that we receive from “outside”.

Try, instead of typing a color in your form, typing a double-quote, a greater-than symbol, and then some text. Something like:

```
"><h1>Your mother smells of elderberries</h1>
```

That wasn't very nice of us to make a web page that says that, was it? In general, you want to be extremely careful about what you accept from outside of your PHP code. Form data can be very easily falsified.

One solution in this case is, as mentioned, keeping a list of valid colors. If a visitor to your site gives a color that is not in your list, you can remember it, but don't use it to modify your page. Checking against a list of colors requires knowing how to use lists in PHP, however, which we haven't gotten to yet. We'll get to that on the next project. But keep this in mind, because web form attacks really start to matter when you start using PHP to send e-mail, as we'll do in the final section on *this* project.

PHP With E-Mail

There is also a function that sends e-mail from PHP. You can take form results and compile them into an e-mail message, and send that e-mail to yourself.

The 'mail' function has three parts: the address you're sending to, the subject of the message, and the body of the message. Add the bold sections below:

```
<?
    $color = $_REQUEST["color"];
    $name = $_REQUEST["name"];
    IF (isset($color)):
        IF ($color == "" || $name == ""):
            echo "<p>You need to enter a color and a name!</p>\n";
        ELSE:
            //send me an e-mail with their favorite
            $subject = "$name's favorite color";
            $sendto = "youraddress@wherever.com";
            $message = "$name said their favorite color was $color.";
            mail($sendto,$subject,$message);

            //display their favorite color
            $color = strtolower($color);
            $color = trim($color);
            IF ($color == "white"):
                $bgcolor = "grey";
            ELSE:
                $bgcolor = "white";
            ENDIF;

?>
    <p style="color: <?echo $color?>; background-color: <?echo $bgcolor?>;">
        You said your favorite color was <?echo $color?>.
    </p>
<?
        ENDIF;
    ELSE:
        echo "<p>Welcome to our color extravaganza!</p>\n";
    ENDIF;
?>

<form method="post" action="test.php">
    <p>
        What is your favorite color?
```

```
        <input type="text" name="color" value="<?echo $color?>" /><br />
        What is your name?
        <input type="text" name="name" value="<?echo $name?>" />
    </p>
    <input type="submit" />
</form>
```

Here, the function (mail) takes more than one "argument". Each item between commas, between the parentheses, is an "argument" to the function. We're sending "mail()"arguments to specify the address the message should go to, the subject of the message, and the body of the message.

Don't forget to replace "youraddress" with your e-mail address!

Finally, notice the two lines that begin with double slashes. PHP ignores any line that begins with double slashes. We can use this to put *comments* in our script. When you have any script larger than a few lines, it is very useful to comment each piece of the script so that you can remember what that piece's purpose is later.

A Note About Email

Never send an e-mail to an address collected on an unprotected form. Remember the color lesson: you can't trust that the person on the other end is honest—or even that they're a person. Don't let your forms become spam sources.

PHP With Files

PHP can also create, append to, and read files on the server. You can use this to store simple data, or to log access information, for long-term retrieval. For example, let's say we want to have a poll and keep track of the results.

Make a new web page with a simple poll in it. Call this file "poll.php".

```
<html>
    <head>
        <title>The Best Imaginary Character</title>
    </head>
    <body>
        <h1>Imaginary Fight to the Finish</h1>
        <form method="post" action="poll.php">
            <p>
                Please choose your favorite imaginary imaginary character:
                <select name="imagine">
                    <option value="">Choose:</option>
                    <option value="rabbit">The White Rabbit</option>
                    <option value="scarecrow">The Scarecrow</option>
                    <option value="tinman">The Tin Man</option>
                    <option value="neo">Neo</option>
                </select>
                <input type="submit" value="Submit your answer" />
            </p>
        </form>
    </body>
</html>
```

This is fairly simple and mostly useless. It doesn't remember anything yet, but it should let you make the choice. Once you have the HTML in the page working, we can go ahead and remember the poll data.

Before you do anything else, you need to create a folder to store your poll data. If you are logged in via your desktop, you can make a folder in your home directory as normal. Call it "data". If you have command line access to your account, you might use:

```
mkdir ~/data
cd ~/data
pwd
```

The 'mkdir' command creates a directory. The 'pwd' command tells you where you currently are. Where the example code says "/path/to/data", replace that with the results of the 'pwd' command. It might well say, for example, "/home/username/data". If so, where it says "/path/to/data/choices.txt" in the example below, use "/home/username/data/choices.txt".

Now that you have the folder for storing the data, add the PHP. At the very top of the page, even before the <html>, add:

```
<?
    $filename = "/path/to/data/choices.txt";
    IF ($choice = $_POST["imagine"]):
        //let's append this to a file
        IF ($choicefile = fopen($filename, "a")):
            fwrite($choicefile, "$choice\n");
            fclose($choicefile);
        ELSE:
            echo "<p><b>Unable to append to file choices.</b></p>\n";
        ENDIF;
    ENDIF;
?>
```

What this should do is, first, open the file called "choices.txt" in the specified directory. The first item in the list is the filename. The second item is what we want to do with the file. In this case, we want to "append" to the file. If we are able to open it, we write the choice and a new line to the file. Then, we close the file.

It is important to close the file as soon as you are done with it, because you can have many different people coming in to view your web page at the same time. Your PHP scripts can be running multiple times all at the same time, but only one of them can write to your file. The rest have to wait until the file is available again.

When you try to use this program, you will almost certainly get an error that looks like:

> Warning : fopen(choices.txt) [function.fopen]: failed to create stream: Permission denied in /home/jerry/public_html/php/poll.php on line 4

This is a typical, and useful, PHP error message. It tells you exactly which line the error occurred on, and it tells you why the error occurred. In this case, "permission" was "denied" for whatever

that line tried to do. Count up to line four in our code, and you'll see that's the line that tries to open the file.

The web server does not have permission to create files. That would be a major security flaw. You will need to create the file by hand, and then you will need to give the web server permission to write to that file.

Go to the command line (in Mac OS X, the 'terminal') and get to the folder where you are storing your PHP files. In Mac OS X, you can easily switch to a folder you're looking at by opening the terminal, typing the letters 'cd' and a space, and then dragging that folder onto the terminal. Go into the terminal and press return to complete the command. Your typing will look something like this:

```
cd ~/data (for example, or drag and drop as described above)
touch choices.txt
chmod ugo+rw choices.txt
```

This will create an empty file called "choices.txt". You can look at it by typing "more choices.txt". Nothing should happen, because it should be empty. But if you then go back to your web page, make a selection, and submit it, you should be able to type "more choices.txt" and see the choice that you made. Keep making choices, and you'll see additional lines show up in the file.

Error Reporting

If you did not receive that error (and if PHP still did not work) it may be that you do not have error reporting turned on. At the very top of your web page, in the php, add:

```
error_reporting(E_ALL ^ E_NOTICE);
```

This turns on all errors, and then turns off notices. See the php.net manual for more information about errors and notices.

Write Access

If you are using PHP to add information to a file in your account, you need to give PHP access to "write to" the file. In general, it is best to create this directory *outside* of your web area.

These two commands will, in Unix, create a folder and then ensure that nobody but you has access to see what is in that folder or make new things in that folder:

```
mkdir ~/data
chmod go-rw ~/data
chmod go+x ~/data
```

This ensures that your data folder is neither readable nor writable by the web server, but that the web server can get in that directory to modify and read files there. Only files that you specifically make readable or writable will be able to be read from or written to by the server.

Before the web server can write to a file, you need to create that file. The usual way to create a file on Unix is to use the 'touch' command, such as:

```
touch ~/data/choices.txt
chmod go+rw ~/data/choices.txt
```

The web server runs as a user (often 'nobody' or 'www') and is part of a group (which may be anything). So if you have particularly sensitive information, you can set your file to be readable and writeable only by the web server's user or group. Look up the 'chmod' command in Unix for more information.

It will usually be best to use either 'g' or 'o' but not both. Only one of them should be necessary, but which one depends on your server.

You can use the 'ls -l' command to see files and their permissions, and the "more" command to see what a file contains.

Whenever you create a file that the web server can write to, you are reducing the security of your system. If someone not on the system can determine the name of that file, and if they can trick your code or someone else's code into writing to that file, they might be able to write a computer program to that file. That computer program now runs as you. So you want to be very careful about not letting people list directories that contain writable files.

Arrays

We'd like to be able to make our poll data available to the public. We can do this by reading our file and collating the data. When you have lots of similar data, you will often store this data in a container called an *array*. For example, if you want to keep a list of colors, you might have an array that contains "red", "blue", "green", "cyan", "yellow", and "magenta". The simplest arrays are just lists of values.

Arrays can also be more complex. For example, you might have a list of employee identification numbers and employee names and phone numbers. If so, you could take the employee identification number and use that to look up an employee's name and the employee's phone number.

What we're going to do is first read the entire file into a simple array that lists every single line in the file. Then we're going to ask PHP to count all of the similar entries. PHP will return that count in a slightly more complex array that corresponds each unique entry with the number of times that entry appears

If fourteen people have voted for the Tin Man and three people have voted for Neo, the simple array will contain fourteen tinmans and three neos. The count will contain "tinman" and correspond that to "14", and one "neo" that corresponds to "3".

Add the following code to the bottom of your web page, after the form but before the </body>:

```
<?
    //get all of our votes
    $votes = file($filename);
    $votecounts = array_count_values($votes);
    FOREACH ($votecounts as $choice => $count):
        echo "<p>$choice received $count vote(s).</p>\n";
    ENDFOREACH;
?>
```

First, we read our file using the "file" command. It takes every line of the file and places each line as an entry in the array $votes. We then use the function array_count_values() to count the items in the $votes array. *That* result is stored in $votecounts.

The *foreach* structure is a lot like the *if* structure. Like *if*, everything between the FOREACH and the ENDFOREACH are performed while the *foreach* is valid. Unlike the *if*, however, the lines enclosed by a foreach block can, and usually are, performed more than once. PHP goes through those lines once for every entry in the array. If there are five items in the array, PHP will use those lines five times. In between the parentheses, we give *foreach* the array and the name of the container(s) that should contain the current array item each time we go through the "loop".

In the $votecounts array, each item is composed of two parts: the choice, and the count of how many times that choice was chosen. So when we say "$votecounts as $choice => $count", we're telling PHP to give us the first part of the array's item in the container $choice and the second part in the container $count. If there were only one part (such as the $votes array), we would use something like "$votes as $choice".

Accessing Array Data

The above code works. It accepts votes and it shows the results of the votes. But the results look pretty ugly. We're showing the viewer our internal representation of the votes, not the human representation. Instead of displaying "The White Rabbit" for example, we're displaying "rabbit". We need to create an array that stores our internal representation and corresponds it to our English representation.

Go to the top of your poll page and, as the first line directly after the "<?", add one line:

```
$pollchoices = array(
    "rabbit"=>"The White Rabbit",
    "scarecrow"=>"The Scarecrow",
    "tinman"=>"The Tin Man",
    "neo"=>"Neo",
);
```

This is an array that corresponds our internal code (such as "rabbit") with the full name of that choice (such as "The White Rabbit"). The left side of each item=>item pair is the "index" of that pair. It can be used to look up the right side of the pair.

Change the inside of our *foreach* to:

```
$name = $pollchoices[$choice];
echo "<p>$name received $count vote(s).</p>\n";
```

You will probably end up seeing something like:

received 1 vote(s).

received 2 vote(s).

received 2 vote(s).

Which is not exactly what we want. The problem is that each of the lines in our file contains a carriage return. When *file()* reads the lines into our array, it does not delete the carriage returns. Our array of real names does not include the carriage returns, and we probably don't want it to. So we need to get rid of the carriage returns. There is a command called "trim()" that does this for us. Change the "$name =" line to:

```
$name = $pollchoices[trim($choice)];
```

You should now see votes with real names:

The Tin Man received 1 vote(s).

Neo received 2 vote(s).

The White Rabbit received 2 vote(s).

Let's say we want to add a new choice. We currently need to do this in *two* places. Once in our form, and once in our $pollchoices array. Now that we have $pollchoices, we can use that to create our form's option lines. Replace all of the option lines with:

```
<option value="">Choose:</option>
<?
    FOREACH ($pollchoices as $code => $name):
        echo "<option value=\"$code\">$name</option>\n";
    ENDFOREACH;
?>
```

Those backslashes ("\") are very important. So far, whenever we've told PHP to display some text, we've surrounded that text with double quotes. But here, one of the things we want PHP to display are double quotes! If you simply put the quotes there along, PHP would get confused and think we wanted to end the text. By "backquoting" the double quotes, PHP knows that we want those quotes displayed, and that they do not end the string of text. (You can also backslash backslashes if you need to display a backslash.)

Finally, go ahead and add ""lion"=>"The Cowardly Lion"" to the list of items in $pollchoices.

Don't Trust Anyone Outside

Remember what we said about trusting data from outside of our PHP code? We're doing it again. They're choosing a selection and we're writing that selection to a file. Right? Not at all. They are

sending us text, and we are writing whatever they send us to a file. If they want to write things to our file other than what is in that list, our form tells them exactly how to do it. That's not good.

Now that we are constructing our selections via a list, we know exactly what options they are authorized to choose. We can check to make sure that the text they sent us is one of the options in our list. Let's add another "if" under the if that sets the "$choice" container:

```
IF ($choice = $_POST["imagine"]):
   //make sure that they sent us a valid choice
   IF ($pollchoices[$choice]):
      //yes, this choice exists
      //let's append this to a file
      IF ($choicefile = fopen($filename, "a")):
         fwrite($choicefile, "$choice\n");
         fclose($choicefile);
      ELSE:
         echo "<p><b>Unable to append to file choices.</b></p>\n";
      ENDIF;
   ENDIF;
ENDIF;
```

Sorting Arrays

You can sort arrays as well, using "asort()" and "arsort()". The first sorts in ascending order, the second in descending (reverse) order. Why not show our results in order from the most popular on down? In front of your foreach where you display the poll's results, add:

```
arsort($votecounts);
```

Our display will now automatically adjust itself according to who is winning the poll.

Visitor Sessions

Our ability to enter the poll as many times as we want has been very useful while testing, but it makes it trivial for anyone to skew our poll to whichever answer they want. All they have to do is keep hitting "reload", or write a script of their own to keep reloading the page.

We can use "cookies" to keep track of individual visitors. PHP can handle this tracking for us almost automatically.

At the top of our page, below $pollchoices, add:

```
session_start();
IF ($existingchoice = $_SESSION["choice"]):
   $alreadychose = true;
ELSE:
   $alreadychose = false;
ENDIF;
```

This starts a session. When you start a session in PHP, PHP automatically checks to see if there is already an existing session on the client. You can store data in the session, using $_SESSION. So here, after we start the session, we check to see if the session data already includes a choice having been made. If it is, we set our container $alreadychose to true. Otherwise, we set it to false. We'll use this later to display either the form or the thanks for voting message.

Change the line that says:

IF ($choice = $_POST["imagine"]):

to

```
IF (!$alreadychose && $choice = $_POST["imagine"]):
    //make sure that they sent us a valid choice
    IF ($pollchoices[$choice]):
        //yes, this choice exists
        $_SESSION["choice"] = $choice;
        $alreadychose = true;
        $existingchoice = $choice;
```

So far we have only checked one thing to see if we should use the PHP inside our "if" areas. Here we are checking two things: first, have they already chosen? The exclamation point reverses that check, so what we're checking is if they have *not* already chosen. Then, if they have not already chosen, we check to see if they have just made a choice. If so, we add that choice to their session. We also have to set $alreadychose and $existingchoice here, so that we thank them rather than show them the form again—remember that they don't get the cookie until we give it to them, and since we only just now gave it to them, $alreadychose and $existingchoice are still blank. We have to set them here.

We now can remember if they have made a choice and what the choice was. Anything we put into $_SESSION will be remembered whenever they return to our page. We aren't going to do it now, but you might remember the last time they visited, for example.

In front of the <form…> line, add:

```
<?IF ($alreadychose):?>
    <p>Thanks for voting for <?echo $pollchoices[$existingchoice]?>!</p>
<?ELSE:?>
```

And, after the </form> line, add:

```
<?ENDIF?>
```

When you try to load this page now, you might see something like:

> Warning : session_start() [function.session-start]: Cannot send session cache limiter - headers already sent (output started at /home/jerry/public_html/php/poll.php:2) in /home/jerry/public_html/php/poll.php on line 4

This means that you did *not* put the PHP code at the very top of your web page. Cookies are not part of your web page's document area. They are part of the "headers" that you rarely see as a user. The headers must be sent before the document is sent. Once you start sending the document, even if it is just a space or a carriage return, you can no longer send any headers. Make sure that the "<?" that starts your PHP code is at the very top of your file.

Requiring Cookies

It's still pretty easy to "beat" our poll. All anyone has to do is turn off cookies in their browser. If they turn off cookies, they'll never send the cookie back to us, and we'll never know that they were already here. You can go ahead and check that right now: find our cookie in your web browser's preferences, delete it, and you can vote once again.

Sessions and Security

Sessions are based on cookies. Cookies are fully under the control of the browser. No matter how much you try to secure your application from cookie manipulation, you cannot win. You cannot rely on cookies being kept.

There are a number of issues with cookies besides that as well. Our poll above will not let anyone vote who uses the same computer as someone who has already voted.

The Program So Far

Here is the full text of our PHP-enhanced web page so far:

```
<?
    $pollchoices = array(
        "rabbit"=>"The White Rabbit",
        "scarecrow"=>"The Scarecrow",
        "tinman"=>"The Tin Man",
        "neo"=>"Neo"
    );

    session_start();
    IF ($existingchoice = $_SESSION["choice"]):
        $alreadychose = true;
    ELSE:
        $alreadychose = false;
    ENDIF;

    $filename = "/path/to/data/choices.txt";
    IF (!$alreadychose && $choice = $_POST["imagine"]):
        //make sure that they sent us a valid choice
        IF ($pollchoices[$choice]):
            //yes, this choice exists
            $_SESSION["choice"] = $choice;
            $alreadychose = true;
            $existingchoice = $choice;

            //let's append this to a file
            IF ($choicefile = fopen($filename, "a")):
                fwrite($choicefile, "$choice\n");
                fclose($choicefile);
```

```
            ELSE:
                echo "<p><b>Unable to append to file choices.</b></p>\n";
            ENDIF;
        ENDIF;
    ENDIF;
?>

<html>
    <head>
        <title>The Best Imaginary Character</title>
    </head>
    <body>
        <h1>Imaginary Fight to the Finish</h1>

        <?IF ($alreadychose):?>
            <p>Thanks for voting for <?echo $pollchoices[$existingchoice]?>!</p>
        <?ELSE:?>

            <form method="post" action="poll.php">
                <p>
                    Please choose your favorite imaginary imaginary character:
                    <select name="imagine">
                        <option value="">Choose:</option>
                        <?
                            FOREACH ($pollchoices as $code => $name):
                                echo "<option value=\"$code\">$name</option>\n";
                            ENDFOREACH;
                        ?>
                    </select>
                    <input type="submit" value="Submit your answer" />
                </p>
            </form>

        <?ENDIF?>

        <?
            //get all of our votes
            $votes = file($filename);
            $votecounts = array_count_values($votes);
            arsort($votecounts);
            FOREACH ($votecounts as $choice => $count):
                $name = $pollchoices[trim($choice)];
                echo "<p>$name received $count vote(s).</p>\n";
            ENDFOREACH;
        ?>

    </body>
</html>
```

Creating Images On the Fly

Some installations of PHP have the ability to create images on the fly. If you need to ask your server administrator, ask if “GD” or “ImageMagick” has been compiled into PHP. These examples assume the “GD” that usually comes built-in to PHP.

Create a new file, call it “graph.php”, and place the following code into it:

```
<?
    $width = $_GET['width'];
    $height = $_GET['height'];
    $image = ImageCreate($width,$height);
```

```
    $color = ImageColorAllocate($image,255,0,0);
    ImageFilledRectangle($image,0,0,$width, $height,$color);
    //send the image
    header("content-type: image/png");
    ImagePNG($image);
?>
```

The $_GET array contains only those $_REQUEST items that came from the URL. So when we call this page, we will need to specify the width and height as http://hostname/path/graph.php?width=xx&height=xx.

The GD image creation function ImageCreate() creates a blank image with the width and height we specify. In order to place color into the image, we need to also create the color, using ImageColorAllocate. We specify colors using three numbers. The numbers range from 0 to 255, and are for red, green, and blue. A high number in red and a low number in green and blue will give a red color, and so on.

We then fill the image with a rectangle that is as big as the image itself, using the red color we've created.

The important part of this is that we're sending a header to tell the web browser that this is not a web page, it's really an image. If you remember from the section on sessions, headers *must* be sent before you send any other data. If you get the "too late to send headers" error, make sure that there is no space or carriage return between the beginning of your document and the "<?" that starts the PHP code.

We're not going to cover it here, but you can use the content-type header to send all sorts of document types. If you want dynamically-modified JavaScript, you can create it using PHP and send it with the content type "text/javascript". Or you can send dynamically-modified style sheet with the content type "text/css". (For JavaScript, you might use "application/x-javascript" instead. This is the technically correct content type. But the HTML 4.0 standard examples use text/javascript, so this is often what browsers support.)

Here we are sending it as "image/png". PNG is the format of the image we are sending. We send our image with the ImagePNG function.

Now, change

```
FOREACH ($votecounts as $choice => $count):
    $name = $pollchoices[trim($choice)];
    echo "<p>$name received $count vote(s).</p>\n";
ENDFOREACH
```

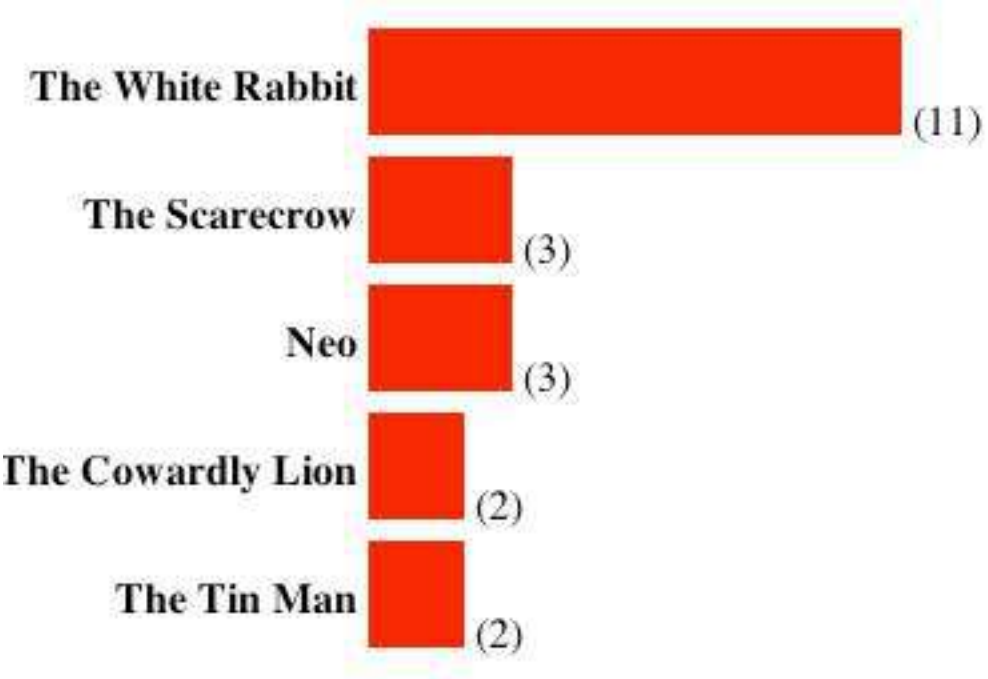
to

```
echo "<table>\n";
FOREACH ($votecounts as $choice => $count):
    $name = $pollchoices[trim($choice)];
    IF (!$maximum):
        $maximum = $count;
    ENDIF;
    $width = $count/$maximum*200;
    echo "<tr><th align=\"right\">$name</th><td>";
```

```
    echo "<img src=\"graph.php?height=40&width=$width\" />";
    echo "($count)</td></tr>\n";
ENDFOREACH;
echo "</table>\n";
```

Our quote marks have to be backquoted so that PHP doesn't think we're ending the "echo" text.

You should get a graph of the vote totals that looks like:



In real life you wouldn't use on-the-fly graphic creation for such a trivial task. You could more easily create a bar image and resize the bar using the <img> tag's "width" option. But if you want to create a line graph or create a pie chart on the fly, you'll find the image creation tools invaluable.

Creating Your Own Functions

We've used several of the built-in functions, and know where to find many more. Sometimes, however, you'll want to create your own functions. You will want to do this when you need to do the same thing in several places on your web page. Instead of typing the same code over and over, you can create a function that you can call at each place where you need those things to happen. This way, if you modify the way that thing happens, you only need to modify it in one place.

At the very end of your web page, even after the </html>, add:

```
<?
    //create a select form item based on the array $options
```

```
    function makeSelect($name, $options) {
        $select = "<select name=\"$name\">\n";
        $select .= "<option value=\"\">Choose:</option>\n";
        FOREACH ($options as $code => $title):
            $select .= "<option value=\"$code\">$title</option>\n";
        ENDFOREACH;
        $select .= "</select>\n";

        return $select;
    }
?>
```

You create your own function with the "function" keyword. After the word "function" you need to type the name of your function. After the name of your function, you place parentheses and the names of any arguments your function requires. These arguments will work just like the arguments for the built-in functions. Here, for example, our function "makeSelect" requires two arguments. Wherever we call "makeSelect", those two arguments will be placed into the containers $name and $options.

The containers within a function are completely separate from any other containers in your page, except for the all-capital underscore containers such as $_REQUEST. Even though our function contains a $name and we already use a $name elsewhere, these two containers are separate and will not interfere with each other. Functions cannot "see" any containers outside of themselves. They can see only what you send them as arguments.

From there, we create a container called $select. We use the ".=" instead of "=" to "append" to our container. The first line puts something into the $select container; the second and subsequent lines append their information to what is already in $select.

Finally, we "return" something to whoever or whatever "called" this function. We return $select, which is the container that contains our <select…> lines.

We can use this function to easily create a <select…> pull-down menu with any name and from any array. Replace the lines that say:

```
<select name="imagine">
    <option value="">Choose:</option>
    <?
        FOREACH ($pollchoices as $code => $name):
            echo '<option value="', $code, '">', $name, '</option>', "\n";
        ENDFOREACH;
    ?>
</select>
```

with:

```
<?echo makeSelect("imagine", $pollchoices)?>
```

You should see exactly what you saw before. But you can now add another array to the beginning of your file:

```
$movies = array("sahara"=>"Sand Pirates of the Sahara",
    "mouse"=>"The Mouse-Trap",
    "rose"=>"The Purple Rose of Cairo",
```

```
    "love"=>"The Crimes of Love"
);
```

And then you can add, directly below your makeSelect that creates the poll choice,

```
<br />
What is your favorite movie? <?echo makeSelect("movie", $movies)?>
```

With a single line, you have added a select for another array. If you decide to change the way that your selects work, you need only change the function to change every select on your page.

Optional Arguments

Sometimes, you'll have two uses for what is basically the same function, but each use has a slightly different argument list. For example, suppose that sometimes we want to specify a default choice for our select form items. The function that supports a default selection will be the same as the function that doesn't support a default extension, with just a little bit of extra PHP code to set the default.

We can combine those two uses into a single function. Change the "function" line of the function to:

```
function makeSelect($name, $options, $default=false) {
```

When you create a function with a $container=something, it means that argument is *optional*. If you don't supply that argument when you call the function, the container will be set to whatever you put after the equal sign.

In this example, $default will be set to *false* if we don't specify a default.

Now, we can break up the <option …> portion of the function, and add a " selected" to the option tag if this $code is the same as the $default we've sent the function.

```
<?
    //create a select form item based on the array $options
    function makeSelect($name, $options, $default=false) {
        $select = "<select name=\"$name\">\n";
        $select .= "<option value=\"\">Choose:</option>\n";
        FOREACH ($options as $code => $title):
            $select .= "<option value=\"$code\"";
            //if there is a default choice, check to see if this is it
            IF ($default):
                IF ($default == $code):
                    $select .= " selected";
                ENDIF;
            ENDIF;
            $select .= ">$title</option>\n";
        ENDFOREACH;
        $select .= "</select>\n";

        return $select;
    }
?>
```

And now you can change “makeSelect("movie", $movies)” to “makeSelect("movie", $movies, $_REQUEST["movie"])”. The movie selection will always default to the previous movie selection.

All optional arguments must be at the end of the argument list. An optional argument before a required argument wouldn’t make sense.

Working With MySQL

PHP works with many databases. One of the more commonly-available databases are MySQL and SQLite. Many hosting services provide MySQL along with PHP. We'll use MySQL here to create a simple database and to store our poll data in MySQL table.

Create the table

The first step is to create the table that will store the data. You'll want to learn to use MySQL, using a tutorial such as my own MySQL for Other Applications.

Our table will include an automatically incremented ID field, a choice field, an IP address field, and a timestamp field. You can create it in a GUI such as CocoaMySQL or by using the mysql command line.

Create the Database

Most likely your system administrator will create the database for you. But if you are running MySQL on your own computer, you'll need to create the database:

```
CREATE DATABASE poll;
```

Create the Username and Password

Again, you'll most likely be given the username by your system administrator. But if you need to create it yourself, you can use something like:

```
GRANT INSERT, SELECT ON poll.* TO polltaker@localhost IDENTIFIED BY
".gE!KAUcuSk8"
```

Create the Table

Finally, you will almost certainly need to create the table yourself. A MySQL table consists of a series of column definitions.

```
CREATE TABLE imaginaries (
    id int(11) unsigned NOT NULL auto_increment PRIMARY KEY,
    choice varchar(30) NOT NULL,
    clientIP varchar(20) NOT NULL,
    timestamp int(11) unsigned NOT NULL
);
```

All of your tables should have an auto increment ID. It makes things a whole lot easier when you start relating one table to another, and when you give users the option of modifying or deleting entries.

The "chloice" is where we'll put the actual choice that the user made.

The clientIP and the timestamp columns will record the IP address that the client came from when they made their choice, and the Unix time that they made their choice. This may be useful for analysis of the data later. It may, for example, be used to check for fraud or ballot-stuffing.

STORE THE DATA

The section that begins with the comment "let's append this to a file" needs to be replaced. Here's the old code:

```
//let's append this to a file
IF ($choicefile = fopen($filename, "a")):
   fwrite($choicefile, "$choice\n");
   fclose($choicefile);
ELSE:
   echo "<p><b>Unable to append to file choices.</b></p>\n";
ENDIF;
```

Also, a little above that, we store the name of the file:

```
$filename = "/path/to/data/choices.txt";
```

So, instead of setting the name of the file, we'll need to set the name of the database and table, and the username and password we'll use to access it. Replace the "$filename=" line with:

```
//table for storing data
$table = "imaginaries";
```

You can see that storing data in a database tends to be initially more complicated than storing it in a file. But it's a whole lot simpler to store data in a database if, as in this case, you want to store more than one piece of information.

Replace the "let's append this" section with:

```
//let's store this in our database
$clientIP = $_SERVER['REMOTE_ADDR'];
$timestamp = time();
$insert = "INSERT INTO $table SET choice='$choice', clientIP='$clientIP',
timestamp=$timestamp";
doQuery($insert);
```

Because we're going to need to perform a query both for inserting and for selecting data, I've placed the query code into its own function:

```
function doQuery($query) {
   $server = "localhost";
   $db = "poll";
   $user = "polltaker";
   $pw = ".gE!KAUcuSk8";
   IF (mysql_connect($server, $user, $pw)):
      IF (mysql_select_db($db)):
         IF (!mysql_query($query)):
            mysql_problem("Unable to perform query $query");
         ENDIF;
      ELSE:
         mysql_problem("Unable to select database $db");
```

```
        ENDIF;
        mysql_close();
    ELSE:
        mysql_problem("Unable to connect to database server $server");
    ENDIF;
}
```

There are more places for things to go wrong here, so we have three different possible errors. Rather than duplicate the same error code each time, I've set the above to use a function called mysql_problem. We need to create that function:

```
function mysql_problem($message) {
    echo "<p><b>$message: ", mysql_error(), "</b></p>\n";
}
```

So, instead of using "fopen" to open the file, we're using "mysql_connect" to connect to the server and "mysql_select_db" to choose the database we'll be storing the information in. Instead of using "fwrite" to write the data to the file, we're using "mysql_query" to insert the data into the database. And instead of using "fclose" to close the file, we're using "mysql_close" to close the database server connection.

Now, when you choose a favorite, the PHP script will store it in the database rather than in the file. You'll notice that if you start testing it here, your vote counts won't increase: that's because the display section of our code is still looking at the file.

If you want to test at this point—and you should—you'll need to look directly in the database using:

```
SELECT * FROM imaginaries;
```

Display the data

The final step, now that we're storing our data in the database is to display data from the database instead of from the file.

One of the nice things about databases is that they usually will sort the information for you. We'll be able to throw out the "array_count_values" and the "arsort" lines. These three lines can go away:

```
$votes = file($filename);
$votecounts = array_count_values($votes);
arsort($votecounts);
```

Replace them with:

```
//get all of our votes
$select = "SELECT choice, COUNT(choice) AS count FROM imaginaries GROUP BY
choice ORDER BY count DESC";
$votecounts = doQuery($select);
```

The tough part there is the select query we're creating. The "GROUP BY choice" part of the query tells MySQL that we only want one row per choice. By asking for the "COUNT(choice)"

we're getting the number of entries for each choice, just as we did originally with "array_count_values".

We need to modify our doQuery function so that it returns the results of any query we send it. Replace:

```
IF (!mysql_query($query)):
   mysql_problem("Unable to perform query $query");
ENDIF;
```

with:

```
IF ($query_result = mysql_query($query)):
   WHILE ($row = mysql_fetch_array($query_result)):
      $rows[] = $row;
   ENDWHILE;
ELSE:
   mysql_problem("Unable to perform query $query");
ENDIF;
```

At the end of the function, add:

```
return $rows;
```

The new mysql function in this version of the doQuery function is mysql_fetch_array. This grabs the next row. So we use "WHILE" to loop through the query result for every row. We're storing it in an array of our own called "rows". When PHP sees "$variable[]=", the empty square brackets tell it to append to that array. This means that at the end of that WHILE loop, we have every row of the query in the array called $rows.

Which means we're pretty much there. Our display loop expects to see a $choice and $count variable, so replace:

```
FOREACH ($votecounts as $choice => $count):
```

with:

```
FOREACH ($votecounts as $votecount):
   $choice = $votecount['choice'];
   $count = $votecount['count'];
```

The keys "choice" and "count" are the names of the columns from our query. The array that mysql_fetch_array created used the column names as the keys for that array.

Curly Brackets

When you look at other people's programs, you'll likely see a lot more curly brackets than we've used in this tutorial. Many programmers, especially those who are familiar with C, Java, Javascript, or Perl, will use curly brackets in PHP to mark the beginning and end of conditional blocks. Instead of IF:/ENDIF they'll use IF { }.

For example, this foreach loop should look familiar to you by now:

```
FOREACH ($items as $item):
   echo "<li>$item</li>\n";
ENDFOREACH;
```

But programmers from other languages might write the same code as:

```
foreach ($items as $item) {
   echo "<li>$item</li>\n";
}
```

I've used the former syntax throughout this tutorial because I find that it is easier—especially for beginners—to see where their conditional blocks end if the ending states what the purpose of that ending is. PHP is often interspersed with HTML. If you see "ENDWHILE;" sitting on its own at the end of some HTML, you know that this is the end of a WHILE: loop. If you see "}" at the end of some HTML, you'll need to work to find the "{" that it matches. It could be the end of an if, a foreach, a while, a for, or a switch.

Some text editors will let you double-click the close curly bracket to match to its opening curly bracket. But because PHP is interspersed with HTML, this doesn't always work as well as it does in programming languages that are not embedded in HTML.

GNU Free Documentation License

Version 1.1, March 2000

0. Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely

available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. Future Revisions of this License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

PHP: Hypertext Processor

PHP is a full programming language that lets you easily add dynamic content to your web pages. If you have a database with data that needs to be displayed on your web pages or edited from your web pages, PHP lets you quickly mock-up a form and form processor. PHP includes support for many databases out of the box.

http://www.hoboes.com/NetLife/PHP/