PHP 101 (Part 2) − Shakespeare's Rose

By Vikram Vaswani and Harish Kamath

Table of Contents

Not What You Expected.....1

Form........2

...And Function.....3

Operating With Extreme Caution.....4

Shakespeare In The Matrix.....5

If Not This, Then What?.....6

Fortune Smiles.....7

A Little Bit Of Logic.....9

Switching Things Around.....10

Submitting To The King.....11

Miscellaneous Notes.....13

Not What You Expected

In last time's lesson, we taught you the basics of PHP variables, and showed you how to add, multiply and concatenate them together. We also introduced you to Her Majesty's most endearing secret agent, and demonstrated just how useful the require() and include() functions are.

Now, you're probably expecting more of the same this week.

Sorry. We're not in the mood.

What we *are* in the mood for, instead, is a little Shakespeare. Read on to find out more.

Form...

Forms have always been one of quickest and easiest ways to add interactivity to your Web site. A form allows you to ask customers if they like your products, casual visitors for comments on your site, and pretty girls for their phone numbers. And PHP can simplify the task of processing the data generated from a Web−based form substantially − as our first example demonstrates.

```
<html> <head> <basefont face="Arial"> </head> <body> <center>
<form method="GET" action="login.php4"> <table cellspacing="5"
cellpadding="5" border="0"> <tr> <td> <font size="−1">So
what's your name?</font> </td> <td align="left"> <input
type="text" name="name" size="10"> </td> </tr> <tr> <td
colspan="2" align="center"> <input type="submit"> </td> </tr>
</table> </form> </center> </body> </html>
```

The most critical line in this entire page is the <FORM> tag

```
<form method="GET" action="./login.php4"> ... </form>
```

As you probably already know, the ACTION attribute of the <FORM> tag specifies the name of the server−side script − "login.php4" in this case − that will process the information entered into the form, while the METHOD attribute specifies the manner in which the information will be passed.

...And Function

Now for the other half of the puzzle − the "login.php4" script. Normally, this script would accept the name entered, check whether the user existed, and either grant or deny access to the site. Since we have yet to cover PHP's conditional statements and logical operators, we're not going to demonstrate that just yet − instead, we're simply going to show you how the data submitted in the first form is injected into "login.php4" and can be used in that file.

Here's "login.php4"

```
<html> <head> <basefont face="Arial"> </head> <body> <center>
<font face="Arial" size="−1"> I wonder if you've heard of
Shakespeare, <? echo $name; ?>. <p> He postulated that a rose
by any other name would smell just as sweet. <p> What do you
think? </font> </center> </body> </html>
```

And when you enter some data into the form [say "bill"] and submit it, here's what you should see:

```
I wonder if you've heard of Shakespeare, bill.

He postulated that a rose by any other name would smell just
as sweet.

What do you think?
```

As you can see, whenever a form is submitted to a PHP script, all variable−value pairs within that form automatically become available for use within the script. In the example above, once the form is submitted, the variable $name is automatically created in the PHP script "login.php4", and is populated with the value entered into the form by the user.

If you were to try doing the same thing in Perl, you'd need to explicitly write code to extract and obtain the values of the variables in the form. By automatically creating and populating the variables for you, PHP simplifies your code and speeds up development time − two reasons why PHP is preferred to Perl when it comes to form processing.

Obviously, PHP also supports the POST method of form submission. All you need to change the METHOD attribute to "POST".

Operating With Extreme Caution

Of course, the example you've just seen is very rudimentary. To really add some punch to it, you need to know how to construct what the geeks call a conditional statement. And the very basis of a conditional statement is comparison − for example, "if this is equal to that, do thus and such".

PHP comes with a bunch of useful operators designed specifically for use in conditional statement. Here's a list:

Assume $delta = 12 and $omega = 9

Operator	What It Means	Expression	Evaluates To
==	is equal to	$delta == $omega	False
!=	is not equal to	$delta != $omega	True
>	is greater than	$delta > $omega	True
<	is less than	$delta < $omega	False
>=	is greater than or equal to	$delta > = $omega	True
<=	is less than or equal to	$delta <= $omega	False

PHP4 also introduces a new operator, which tests both for equality and type: the === operator. There's a simple illustration of this on the last page.

Shakespeare In The Matrix

In PHP, the simplest form of conditional statement is the "if" statement, which looks something like this:

```
if (condition) { do this! }
```

The "condition" here is a conditional expression, which evaluates to either true or false. If the statement evaluates to true, all PHP code within the curly braces is executed; if not, the code within the curly braces is skipped and the lines following the "if" construct are executed.

Let's show you how the "if" statement works by adding a little user authentication to the "login.php4" script above, such that access will be granted only when the user enters the name "neo".

```
<html> <head> <basefont face="Arial"> </head> <body> <center>
<? // check name and print appropriate message if ($name ==
"neo") { ?> <font face="Arial" size="−1"> Welcome to the
Matrix, Neo. <p> May The Force be with you...oops, wrong
movie! </font> <? } ?> <? // if password is wrong if ($name !=
"neo") { ?> <font face="Arial" size="−1"> I wonder if you've
heard of Shakespeare, <? echo $name; ?>. <p> He postulated
that a rose by any other name would smell just as sweet. <p>
Unfortunately for you, I disagree. Access denied. </font> <? }
?> </center> </body> </html>
```

In this case, the two "if" statements present an appropriate message, depending on whether or not the right password was entered. Of course, PHP also lets you "nest" conditional statements − for example, this is perfectly valid PHP code:

```
<? if ($day == "Thursday") { if ($time == "12") { if ($place
== "Italy") { $lunch = "pasta"; } } } ?>
```

If Not This, Then What?

In addition to the "if" statement, PHP also offers the "if−else" construct, used to define a block of code that gets executed when the conditional expression in the "if" statement evaluates as false.

The "if−else" construct looks like this:

```
if (condition) { do this! } else { do this! }
```

As you can see, this construct can be used to great effect in the last example − instead of two separate "if" statements, we can combine them into a single "if−else" statement.

```
<html> <head> <basefont face="Arial"> </head> <body> <center>
<? // check name and print appropriate message if ($name ==
"neo") { ?> <font face="Arial" size="−1"> Welcome to the
Matrix, Neo. <p> May The Force be with you...oops, wrong
movie! </font> <? } else { ?> <font face="Arial" size="−1"> I
wonder if you've heard of Shakespeare, <? echo $name; ?>. <p>
He postulated that a rose by any other name would smell just
as sweet. <p> Unfortunately for you, I disagree. Access
denied. </font> <? } ?> </center> </body> </html>
```

Fortune Smiles

PHP also provides you with a way of handling multiple possibilities − the "if−elseif−else" construct. A typical "if−elseif−else" statement block would look like this:

```
if (first condition is true) { do this! } elseif (second
condition is true) { do this! } elseif (third condition is
true) { do this! } ... and so on ... else { do this! }
```

And here's an example that demonstrates how to use it:

```
<html> <head> <style type="text/css"> td {font−family: Arial;}
</style> </head> <body> <font face="Arial" size="+2"> The
Amazing Fortune Cookie Generator </font> <form method="GET"
action="cookie.php4"> <table cellspacing="5" cellpadding="5"
border="0"> <tr> <td align="center"> Pick a day </td> <td
align="right"> <select name="day"> <option
value="Monday">Monday <option value="Tuesday">Tuesday <option
value="Wednesday">Wednesday <option value="Thursday">Thursday
<option value="Friday">Friday <option
value="Saturday">Saturday <option value="Sunday">Sunday
</select> </td> </tr> <tr> <tr> <td colspan="2"
align="center"> <input type="submit" value="Hit me!"> </td>
</tr> </table> </form> </body> </html>
```

As you can see, this is simply a form which allows you to pick a day of the week. The real work is done by the PHP script "cookie.php4"

```
<? if ($day == "Monday") { $fortune = "Never make anything
simple and efficient when a way can be found to make it
complex and wonderful."; } elseif ($day == "Tuesday") {
$fortune = "Life is a game of bridge −− and you've just been
finessed."; } elseif ($day == "Wednesday") { $fortune = "What
sane person could live in this world and not be crazy?"; }
elseif ($day == "Thursday") { $fortune = "Don't get mad, get
interest."; } elseif ($day == "Friday") { $fortune = "Just go
with the flow control, roll with the crunches, and, when you
get a prompt, type like hell."; } else { $fortune = "Sorry,
closed on the weekend"; } ?> <html> <head> <basefont
face="Arial"> </head> <body> Here is your fortune for <? echo
$day; ?>: <br> <b><? echo $fortune; ?></b> </body> </html>
```

In this case, we've used the "if−elseif−else" control structure to assign a different fortune to each day.

There's one important point to be noted here − as soon as one of the "if" statements within the block is found to be true, PHP will execute the corresponding code, skip the remaining "if" statements in the block, and jump immediately to the lines following the entire "if−elseif−else" block.

A Little Bit Of Logic

Now, you've already seen that PHP allows you to nest conditional statements. However, if you take another look at the example we used to demonstrate the concept

```
<? if ($day == "Thursday") { if ($time == "12") { if ($place
== "Italy") { $lunch = "pasta"; } } } ?>
```

you'll agree that is both complex and frightening. And so, in addition to the comparison operators we've used so liberally thus far, PHP also provides a few logical operators which allow you to group conditional expressions together. The following table should make this clearer.

Operator	What It Means	Example	Evaluates To
	AND	$delta == $gamma &$delta > $omega	True
		$delta &$omega < $omega	False
||	OR	$delta == $gamma || $delta < $omega	True
		$delta > $gamma || $delta < $omega	False
!	NOT	!$delta	False
<=	is less than or equal to	$delta <= $omega	False

Given this knowledge, it's a simple matter to rewrite the example above in terms of logical operators:

```
<? if ($day == "Thursday" &$time == "12" &$place == "Italy") {
$lunch = "pasta"; }
```

Simple and elegant? Yes.

Switching Things Around

An alternative to the "if−else" family of control structures is PHP's "switch" statement, which does almost the same thing. It looks like this

```
switch (decision−variable) { case first_condition_is true: do
this! case second_condition_is true: do this! case
third_condition_is true: do this! ... and so on... }
```

We'll make this a little clearer by re−writing our fortune cookie example in terms of the "switch" statement.

[cookie.php4]

```
<? // the decision variable here is the day chosen by the user
switch ($day) { // first case case "Monday": $fortune = "Never
make anything simple and efficient when a way can be found to
make it complex and wonderful."; break; // second case case
"Tuesday": $fortune = "Life is a game of bridge −− and you've
just been finessed."; break; case "Wednesday": $fortune =
"What sane person could live in this world and not be crazy?";
break; case "Thursday": $fortune = "Don't get mad, get
interest."; break; case "Friday": $fortune = "Just go with the
flow control, roll with the crunches, and, when you get a
prompt, type like hell."; break; // if none of them match...
default: $fortune = "Sorry, closed on the weekend"; break; }
?> <html> <head> <basefont face="Arial"> </head> <body> Here
is your fortune for <? echo $day; ?>: <br> <b><? echo
$fortune; ?></b> </body> </html>
```

There are a couple of important keywords here: the "break" keyword is used to break out of the "switch" statement block and move immediately to the lines following it, while the "default" keyword is used to execute a default set of statements when the variable passed to "switch" does not satisfy any of the conditions listed within the block.

Submitting To The King

You'll have noticed that in all the examples we've shown you thus far, we've used two pages − a single HTML page containing the form, and a separate PHP script which processes the form input and generates appropriate output. However, PHP provides an elegant method to combine those two pages into one via the $submit variable.

You've already seen that once a form is submitted to a PHP script, all the form variables become available to PHP. Now, in addition to the user−defined variables, each time you hit the SUBMIT button on a form, PHP creates a variable named $submit. And by testing for the presence or absence of this variable, a clever PHP programmer can use a single PHP script to generate both the initial form and the output after it has been submitted.

Let's demonstrate this to you − we've rewritten the fortune cookie example above to use a single PHP file to generate both the initial drop−down list, and the subsequent fortune cookie page. We're assuming that the PHP file is named "cookie.php4"

```
<? if (!$submit) { // if $submit doesn't exist, it implies
that the form // has not yet been submitted // so display the
first page ?> <html> <head> <style type="text/css"> td
{font−family: Arial;} </style> </head> <body> <font
face="Arial" size="+2"> The Amazing Fortune Cookie Generator
</font> <form method="GET" action="cookie.php4"> <table
cellspacing="5" cellpadding="5" border="0"> <tr> <td
align="center"> Pick a day </td> <td align="right"> <select
name="day"> <option value="Monday">Monday <option
value="Tuesday">Tuesday <option value="Wednesday">Wednesday
<option value="Thursday">Thursday <option
value="Friday">Friday <option value="Saturday">Saturday
<option value="Sunday">Sunday </select> </td> </tr> <tr> <tr>
<td colspan="2" align="center"> <input type="submit"
name="submit" value="Hit me!"> </td> </tr> </table> </form>
</body> </html> <? } else { // if $submit does exist, the form
has been submitted // so process it with switch() // the
decision variable here is the day chosen by the user switch
($day) { // first case case "Monday": $fortune = "Never make
anything simple and efficient when a way can be found to make
it complex and wonderful."; break; // second case case
"Tuesday": $fortune = "Life is a game of bridge −− and you've
just been finessed."; break; case "Wednesday": $fortune =
"What sane person could live in this world and not be crazy?";
break; case "Thursday": $fortune = "Don't get mad, get
interest."; break; case "Friday": $fortune = "Just go with the
flow control, roll with the crunches, and, when you get a
prompt, type like hell."; break; // if none of them match...
default: $fortune = "Sorry, closed on the weekend"; break; }
?> <html> <head> <basefont face="Arial"> </head> <body> Here
```

```
is your fortune for <? echo $day; ?>: <br> <b><? echo
$fortune; ?></b> </body> </html> <? }
```

As you can see, the script first tests for the presence of the $submit variable − if it doesn't find it, it assumes that the form has yet to be submitted and so displays the initial list of days.

Since the ACTION attribute of the <FORM> tag references the same PHP script, once the form has been submitted, the same script will be called to process the form input. This time, however, the $submit variable will exist, and so PHP will not display the initial page, but rather the page which contains the fortune cookie.

Note that for this to work, your

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

must have a NAME attribute with the value "submit", like this:

```
<input type="submit" name="submit">
```

Miscellaneous Notes

The === operator

As we've mentioned above, PHP4 introduces the new === operator to test whether variables are of the same type. Here's an example:

```
<? if (!$submit) { // if $submit doesn't exist, it implies
that the form // has not yet been submitted // so display the
first page ?> <html> <head> <style type="text/css"> td
{font-family: Arial;} </style> </head> <body> <form
method="GET" action="cookie.php4"> <table cellspacing="5"
cellpadding="5" border="0> <tr> <td align="center"> Gimme
something! </td> <td align="right"> <input type="text"
name="yin"> </td> </tr> <tr> <td align="center"> Gimme
something else! </td> <td align="right"> <input type="text"
name="yang"> </td> </tr> <tr> <tr> <td colspan="2"
align="center"> <input type="submit" name="submit"
value="Test!"> </td> </tr> </table> </form> </body> </html> <?
} else { // if $submit does exist, the form has been submitted
// so process it if ($yin === $yang) { $result = "Both
variables are identical and of the same type"; } else {
$result = "The variables are either not identical or not of
the same type"; } ?> <html> <head> <basefont face="Arial">
</head> <body> <b><? echo $result; ?></b> </body> </html> <? }
?>
```

Alternative syntax

PHP also supports an alternative syntax for the various control structures discussed so far. For example, you could do this:

```
<? if ($elvis == 0) { echo "Elvis has left the building!"; }
else { echo "Elvis is still backstage!"; } ?> or you could do
this <? if ($elvis == 0): echo "Elvis has left the building!";
else: echo "Elvis is still backstage!"; endif; ?>
```

The second alternative is equivalent to the first, and simply involves replacing the first curly brace of every pair with a colon [:], removing the second curly brace, and ending the block with an "endif" statement.

And that's it for this week. Next time, we'll be bringing you loops, arrays and more forms − so make sure you don't miss it!