PHP 101 (part 2): Calling All Operators

Not What You Expected

In Part One of this series, I gave you a

brief introduction to PHP, and how it fits into your Web application development environment. I also taught you the basics of PHP variables, and showed you how to add, multiply and concatenate them together.

Now that you know the basics, it’s time to focus in on one of PHP’s nicer features – its ability to automatically receive user input from a Web form and convert it into PHP variables. If you’re used to writing Perl code to retrieve form values in your CGI scripts, PHP’s simpler approach is going to make you weep with joy. So get that handkerchief out, and scroll on down.

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 this first example demonstrates.

This example contains two scripts, one containing an HTML form (named form.htm) and the other containing the form processing logic (message.php). Here’s form.htm:

```
<html>
```

```
<head></head>

<body>


<form action="message.php" method="post">

Enter your message: <input type="text" name="msg" size="30">



<input type="submit" value="Send">



</form>



</body>



</html>
```

The critical line in this page is the `<form>` tag

```
<form action="message.php" method="post">


...

</form>
```

As you probably already know, the “action” attribute of the `<form>` tag specifies the name of the server-side script (`message.php` in this case) that will process the information entered into the form. The “method” attribute specifies *how* the information will be passed.

...And Function

Now for the other half of the puzzle: the `message.php` script. This script reads the data submitted by the user and “does something with it”. Here is `message.php`:

```
<html>

<head></head>
<body>
<?php

// retrieve form data

$input = $_POST['msg'];

// use it
```

```
echo "You said: <i>$input</i>";
?>
</body>


</html>
```

When you enter some data into `form.htm` (let’s say “Boo”), and submit it, the form processor `message.php` will read it and display it to you (“You said: *Boo*“). Thus, whenever a form is submitted to a PHP script, all variable-value pairs within that form automatically become available for use within the script, through a special PHP container(可以理解為數組或 map) variable: `$_POST`.

You can then access the value of the form variable by using its “name” inside the `$_POST` container, as I did in the script above.

Obviously, PHP also supports the GET method of form submission. All you need to do is change the “method” attribute to “get”, and retrieve values from `$_GET` instead of `$_POST`. The

`$_GET` and `$_POST` variables are

actually a special type of PHP animal called an array, which I’ll be teaching you about shortly. Don’t worry too much about it at the moment, just make sure you’re comfortable with retrieving simple values from a form with PHP, and then scroll on down to learn about some more operators that are useful in this context.

Operating With Extreme Caution

Thus far, the scripts we’ve discussed have been pretty dumb. All they’ve done is add numbers and strings, and read back to you the data you typed in yourself – not exactly overwhelming. In order to add some intelligence to your scripts, you need to know how to construct what geeks call a “conditional statement” – a statement which lets your script perform one of a series of possible actions based on the result of a comparison test. And since the basis of a conditional statement is comparison, you first need to know how to compare two variables and determine whether they’re identical or different.

You’ve already seen some of PHP’s arithmetic and string operators. However, the language also comes with operators designed specifically to compare two values: the so-called “comparison operators”. Here’s an example that demonstrates them in action:

```
<?php
/* define some variables */
$mean = 9;
$median = 10;

$mode = 9;
// less-than operator
// returns true if left side is less than right

// returns true here

$result = ($mean < $median);
print "result is $result<br />";
// greater-than operator

// returns true if left side is greater than right

// returns false here

$result = ($mean > $median);

print "result is $result<br />";
// less-than-or-equal-to operator

// returns true if left side is less than or equal to right

// returns false here

$result = ($median <= $mode);

print "result is $result<br />";
// greater-than-or-equal-to operator
// returns true if left side is greater than or equal to right
// returns true here

$result = ($median >= $mode);

print "result is $result<br />";
// equality operator

// returns true if left side is equal to right

// returns true here
$result = ($mean == $mode);

print "result is $result<br />";
// not-equal-to operator

// returns true if left side is not equal to right

// returns false here

$result = ($mean != $mode);
print "result is $result<br />";
// inequality operator

// returns true if left side is not equal to right
// returns false here

$result = ($mean <> $mode);   <>相當於!=

print "result is $result";
?>
```

The result of a comparison test is always Boolean: either true (1) or false (0 – which doesn’t print anything). This makes comparison operators an indispensable part of your toolkit, as you can use them in combination with a conditional statement to send a script down any of its multiple action paths.

PHP 4.0 also introduced a *new* comparison operator, which allows you to test both for equality and type: the **===** operator. The following example demonstrates it:

```
<?php
/* define two variables */
$str = '10';

$int = 10;
/* returns true, since both variables contain the same value */

$result = ($str == $int);
```

```
print "result is $result<br />";
/* returns false, since the variables are not of the same type even though they
have the same value */

$result = ($str === $int);

print "result is $result<br />";
/* returns true, since the variables are the same type and value */

$anotherInt = 10;

$result = ($anotherInt === $int);

print "result is $result";
?>
```

Read more about PHP’s comparison operators at
http://www.php.net/manual/en/language.operators.comparison.php.

A Question of Logic

In addition to the comparison operators I used so liberally above, PHP also provides four logical operators, which are designed to group conditional expressions together. These four operators – logical AND, logical OR, logical XOR and logical NOT – are illustrated in the following example:

```php
<?php

/* define some variables */

$auth = 1;


$status = 1;


$role = 4;

/* logical AND returns true if all conditions are true */


// returns true


$result = (($auth == 1) && ($status != 0));


print "result is $result<br />";

/* logical OR returns true if any condition is true */


// returns true


$result = (($status == 1) || ($role <= 2));


print "result is $result<br />";

/* logical NOT returns true if the condition is false and vice-versa */


// returns false


$result = !($status == 1);
```

```
print "result is $result<br />";
/* logical XOR returns true if either of two conditions are true, or returns false if
both conditions are true */

// returns false
$result = (($status == 1) xor ($auth == 1));

print "result is $result<br />";
?>
```

Logical operators play an important role in building conditional statements, as they can be used to link together related conditions simply and elegantly. View more examples of how they can be used at http://www.php.net/manual/en/language.operators.logical.php.

Older But Not Wiser

Now that you’ve learnt all about comparison and logical operators, I can teach you about conditional statements. As noted earlier, a conditional statement allows you to test whether a specific condition is true or false, and perform different actions on the basis of the result. In PHP, the simplest form of conditional statement is the `if()` statement, which looks something like this:

```
if (condition) {

    do this!

}
```

The argument to `if()` 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 it does not, the code within the curly braces is skipped and the lines following the `if()` construct are executed.

Let me show you how the `if()` statement works by combining it with a form. In this example, the user is asked to enter his or her age.

```
<html>

<head></head>
<body>

<form action="ageist.php" method="post">

Enter your age: <input name="age" size="2">
</form>
```

```
</body>

</html>
```

Depending on whether the entered age is above or below 21, a different message is displayed by the `ageist.php` script:

```
<html>

<head></head>
<body>
<?php

// retrieve form data

$age = $_POST['age'];

// check entered value and branch
if ($age >= 21) {

    echo 'Come on in, we have alcohol and music awaiting you!';
}

if ($age < 21) {

    echo "You're too young for this club, come back when you're a little older";
}

?>
</body>

</html>
```

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!

}
```

This construct can be used to great effect in the last example: we can combine the two separate `if()` statements into a single `if-else` statement.

```
<html>
<head></head>
<body>
<?php

// retrieve form data

$age = $_POST['age'];

// check entered value and branch

if ($age >= 21) {
        echo 'Come on in, we have alcohol and music awaiting you!';

    }

else {

    echo "You're too young for this club, come back when you're a little older";

}

?>
</body>

</html>
```

Spreading Confusion

If the thought of confusing people who read your code makes you feel warm and tingly, you’re going to love the ternary operator, represented by a question mark (?). This operator, which lets you make your conditional statements almost unintelligible, provides shortcut syntax for creating a single-statement `if-else` block. So, while you could do this:

```
<?php
```

```
if ($numTries > 10) {
        $msg = 'Blocking your account...';

    }

else {

    $msg = 'Welcome!';

}
?>
```

You could also do this, which is equivalent (and a lot more fun):

```
<?php
? : 運算符:
$msg = $numTries > 10 ? 'Blocking your account...' : 'Welcome!';
?>
```

PHP also lets you “nest” conditional statements inside each other.

For example, this is perfectly valid PHP code:

```
<?php
if ($day == 'Thursday') {
        if ($time == '0800') {

        if ($country == 'UK') {

            $meal = 'bacon and eggs';

        }

    }

}
?>
```

Another, more elegant way to write the above is with a series of logical operators:

```
<?php
if ($day == 'Thursday' && $time == '0800' && $country == 'UK') {
        $meal = 'bacon and eggs';

}
```

```
?>
```

The Daily Special

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></head>
<body>


<h2>Today's Special</h2>
<p>


<form method="get" action="cooking.php">
```

```
<select name="day">

<option value="1">Monday/Wednesday

<option value="2">Tuesday/Thursday

<option value="3">Friday/Sunday

<option value="4">Saturday

</select>

<input type="submit" value="Send">
</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 `cooking.php`:

```
<html>
```

```
<head></head>

<body>

<?php


// get form selection


$day = $_GET['day'];


// check value and select appropriate item


if ($day == 1) {
        $special = 'Chicken in oyster sauce';


    }


elseif ($day == 2) {


    $special = 'French onion soup';


    }


elseif ($day == 3) {
        $special = 'Pork chops with mashed potatoes and green salad';


    }
```

```
else {

    $special = 'Fish and chips';

}

?>
<h2>Today's special is:</h2>
<?php echo $special; ?>

</body>


</html>
```

In this case, I’ve used the `if-elseif-else` control structure to assign a different menu special to each combination of days. Note that as soon as one of the `if()` branches 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.

And that’s it for now. To view more examples of conditional statements in action, visit http://www.php.net/manual/en/language.control-structures.php. In

Part Three, I’ll be bringing you more control
structures, more operators and more strange and wacky scripts – so make sure
you don’t miss it!