Programming via PHP Numbers and strings

5.1. Operators

We have already seen how to use functions, such as `sqrt`, to manipulate values in PHP. For the most common operations, though, PHP provides symbolic shortcuts for the operations, such as `+` for adding two values together, or `*` for multiplying two values. Such symbols are called **operators**.

PHP includes about 30 operators, though we'll only cover the 10 or so that appear frequently in programs. The most recognizable would be `+` for addition, `-` for subtraction or negation, `*` for multiplication, and `/` for division. We can use these to enhance our range selection page so that it displays the midpoint in the range:

```
<p>From the range <?php echo $form_begin; ?>
to <?php echo $form_end; ?> I have selected the random number
<?php echo rand($form_begin, $form_end); ?>.  The midpoint is <?php
    $average = ($form_begin + $form_end) / 2;
    echo $average;
?>.</p>
```

PHP observes the traditional order of operations: Negation comes first, followed by division and multiplication in left-to-right order, followed by addition and subtraction in left-to-right order. You can use parentheses when you want to indicate a different order of operations; in the above example, we had to use the parentheses; otherwise, PHP would add `form_begin` to half of `$form_end`, which wouldn't be the average of the two values.

There is another operator worth mentioning: The **remainder operator**, represented by the percent sign. When placed between two numbers, the resulting value is whatever remainder results when the first number is divided by the second number; for example, `28 % 9` would yield 1, since 9 goes into 28 three times with a remainder of 1. The remainder operator lies at the same precedence level as multiplication and division in PHP.

Note that one operator that is *not* on the list: There is no operator for exponentiation. People often expect the caret ('^') to do this; PHP allows you to use a caret to operator on two numbers, but what it does is fairly obscure and rarely very useful. (You don't need to worry about it, but what it does is a *bitwise exclusive or*, where it computes the number whose binary representation is 1 in each place where the two numbers' binary representations are different.) When you want to raise one number to another power, you

can instead use PHP's `pow` function: The tenth power of 1.5, for example, can be found with `pow(1.5, 10)`.

5.2. Types

As PHP processes a program, it works with a variety of categories of information, each called a **type**. Thus far, we have glossed over the distinction, but it is an important one.

In fact, PHP works with two different types of numbers: **integers** and **floating-point numbers**. Integers are numbers represented in a format that allow no fractional parts (and allow numbers such as 0 or 2 or −158), whereas floating-point numbers are represented in a format that allows fraction parts (such as 3.14159 and 1.5). Whenever a number is written with a decimal point, it is a floating-point number; thus, if you write `1.0`, you're talking about the floating-point value 1, which doesn't behave exactly the same as the integer value 1.

For those who have programmed in other languages based on C: PHP's division operator always produces a floating-point result. (Most other C-derived languages ape C's rules for dividing two integers, where the remainder is ignored so the result is still an integer. In PHP, an expression such as `1 / 2` actually yields 0.5 as a beginner would expect.)

Most of the time you don't need to worry about the distinction between floating-point numbers and integers; PHP freely converts integers to floating-point numbers when necessary. One situation where it can appear is with rounding problems, such as with the following statement.

```
echo 10000000000000000 - 9999999999999999;
```

This statement outputs 0 on some computers, because those computers can't store integers so big, and so PHP uses floats instead... and it happens that they are both approximated with the same float.

If there's some reason you want to convert between the two, you can use PHP's `intval` and `floatval` functions.

5.3. Strings

PHP includes many other types for values that aren't numbers. One of the most important such is the **string**, which is a value representing a sequence of several characters, such as a word, a sentence, or a nonsensical stream of punctuation.

To refer to a particular string value, you can use a set of quotation marks enclosing the characters you want to include in the string. The following statement makes the variable `$sentence` refer to a famed novel's opening sentence.

```
$name = "It was the best of times; it was the worst of times.";
```

When you enclose a string in quotation marks, the contents of the quotation marks are not interpreted by PHP: It simply places the raw characters into the string. Suppose I type:

```
$expr = "rand(1, 100)";
```

PHP does not use the `rand` function here: It simply creates a string consisting of 12 characters, starting with the letter *r* and ending with the close-parenthesis character. If we omitted the quotation marks, of course PHP would call the `rand` function to retrieve a random number between 1 and 100.

One exception (and the only exception) to this rule is variable names: When you include a variable name in a string, PHP will substitute the variable's value in place of the name. Suppose I write this:

```
echo "The range is $form_begin to $form_end. ";
echo "The midpoint is ($form_begin + $form_end) / 2. ";
echo "The chosen number is rand($form_begin, $form_end). ";
```

The PHP interpreter would substitute the values of the variables whenever they occur; but it would not attempt to apply the operators or call the functions. What we would see on the Web browser, then, is the following.

```
The range is 10 to 20. The midpoint is (10 + 20) / 2. The chosen number is rand(10, 20).
```

Of course, that's probably not what we want. There are several ways of getting around this. One is to first assign a variable to refer to the desired value, and then to use that variable inside the string. But another approach worth knowing about is the **catenation operator** '.', which you can use to join two values into a string. The following is how we could modify our example.

```
echo "The range is $form_begin to $form_end. ";
echo "The midpoint is " . (($form_begin + $form_end) / 2) . ". ";
echo "The chosen number is " . rand($form_begin, $form_end) . ". ";
```

As an example, the final line says to catenate the string The chosen number is with the value generated by `rand`, which itself should be catenated with the string .. Note the space that has carefully been added inside the first string after *is*: Without this, the output would omit the space, as in The chosen number is17..

The catenation operator has the same priority level as addition and subtraction. As a result, the following would be incorrect.

```
echo "The sum is " . $form_begin + $form_end;   // WRONG!
```

PHP would first catenate The sum is with `$form_begin`, resulting in a string such as The sum is 20. Then it would attempt to interpret this as a number so that it could add `$form_end` to it. It would not work as we would expect. For this reason, I recommend developing the habit of using parentheses whenever you try to combine other operators with catenation, even when it is not necessary. The above midpoint example illustrates this: Even though there the parentheses were unnecessary since division occurs at a higher level than catenation, I went ahead and used parentheses anyway.

We've seen that variable names in double quotation marks will be replaced with the variable values. But that brings up a question: What if you don't want this substitution to occur? There are two answers that PHP provides. First, you can prefix the relevant dollar sign with a backslash, as in `"\$choice is $choice"`; in this example, the first dollar sign won't be treated as part of a variable name, but the second will, so the string will translate to something like `$choice is 17`. You can do the same backslash trick when you want to a quotation mark to appear inside the string, or when you want a backslash as part of the string.

Backslashes can also be used to insert other characters. Most significantly, '`\n`' inserts a **newline character** into a string. Since line breaks aren't automatically inserted between **`echo`** statement, a PHP script that echoes a large amount of information would normally create one large line, which can be quite difficult to read for somebody who tries to read the generated HTML source. (Of course, the browser ignores line breaks anyway.) For this reason, I'd be inclined to instead write our earlier example with three echo statements as the following instead.

```
echo "The range is $form_begin to $form_end.\n";
echo "The midpoint is " . (($form_begin + $form_end) / 2) . ".\n";
echo "The chosen number is " . rand($form_begin, $form_end) . ".\n";
```

Note the '`\n`' that follows each period. This leads the generated HTML to include three separate lines rather than one long line. Again, the browser will display the HTML the same

when we have spaces instead of newlines, but a user who chooses to read the HTML source will have an easier time with the newlines inserted. (You can also use '`\t`' to specify a tab character, but this is not as useful.)

Sometimes the number of backslashes can get a bit cumbersome, so PHP provides another way, too: You can enclose the string in single quotation marks. Variable substitution never occurs for a string enclosed in single quotation marks. (I won't use this single-quotation technique, but many PHP programmers use single quotes as a matter of habit.)

Source: http://www.toves.org/books/php/ch05-types/index.html