5.4 Advanced PHP Syntax

In these sections we'll see more advanced PHP syntax for elements such as functions, arrays, and files. These will allow us to create larger, well-structured programs that interact with interesting data.

5.4.1 Functions

Like functions and methods in C, C++, and Java, a PHP *function* is a named group of statements that can be executed many times. PHP functions can accept parameters, which are declared by writing their names with dollar signs in front; no types are declared. Example 5.33 shows the syntax.

PHP functions can also return values. To return a value from a function, place a `return` statement in the function's body, using the syntax shown in Example 5.34. For example, the function shown in Example 5.35 computes the slope of a line based on two points (x_1, y_1), (x_2, y_2).

```
function name($parameterName, ..., $parameterName) {
  statements;
}
```

Example 5.33 Syntax template for function declaration

```
  return value;
```

Example 5.34 Syntax template for `return` statement

```
function slope($x1, $y1, $x2, $y2) {
  return ($y2 - $y1) / ($x2 - $x1);
}
```

Example 5.35 Function declaration

As we've already seen earlier in this chapter, functions are called by writing the function's name, and the actual parameter values between parentheses separated by commas. For example, to call the `quadratic` function we just declared, you could write the code shown in Example 5.36.

```
$var1 = -2;
$var2 = 3;
$root = quadratic(1, $var1, $var2 - 2);    # $root stores 1
```

Example 5.36 Function call

You can specify default parameter values to make a function's parameter optional. The syntax for doing so is to declare the parameter with an equals sign followed by the default value to use if none is passed. Any parameters with default values must appear at the end of the list of parameters. The general syntax is shown in Example 5.37, and a function utilizing the syntax is shown in Example 5.38. The `print_separated` function uses a comma and space as the default separator string when printing its output if the caller passes only one parameter.

```
function name(..., $parameterName = value, $parameterName = value) {
  statements;
}
```

Example 5.37 Syntax template for default parameter values

```
function print_separated($str, $separator = ", ") {
  if (strlen($str) > 0) {
    print $str[0];
    for ($i = 1; $i < strlen($str); $i++) {
      print $sep . $str[$i];
    }
  }
}
```

Example 5.38 Function with default parameter value

You can call the `print_separated` function two ways: passing one parameter for a string, in which case its letters will be printed separated by commas and spaces; or passing a second parameter for the text to place between the string's characters. Example 5.39 demonstrates both kinds of calls.

```
print_separated("hello");        # h, e, l, l, o
print_separated("hello", "-");   # h-e-l-l-o
```

Example 5.39 Function calls using default parameter values

Value vs. Reference Parameters

Normally parameters are passed by value in PHP, meaning that the actual parameter values passed are copied into the function's formal parameters during each call. The impact of this is that a function cannot change the values of any variables that are passed in to it. Notice that the code in the function in Example 5.40 is not able to modify the value of `$x` from the main program; `$num`'s value is just a copy of `$x`'s value. They are not linked to each other, so `$x` is still 5 after the call.

```
function make_bigger($num) {
  $num = $num * 2;
}

$x = 5;
make_bigger($x);
print $x;              # 5
```

Example 5.40 Value parameter

Parameters can also be *passed by reference*, which causes the function's parameter to be an alias or link to the original parameter passed from the main program. This means that if the function changes its parameter value, the caller will also see the change in its own parameter. A reference parameter is specified by placing a `&` before the `$` in front of its name. In Example 5.41, parameter `$num` is a reference, meaning that when it is doubled, `$x` passed from the main program also doubles.

```
function make_bigger(&$num) {
  $num = $num * 2;
}

$x = 5;
make_bigger($x);
print $x;              # 10
```

Example 5.41 Reference parameter

Reference parameters can be very useful in some situations. However, overusing them can lead to abuse of them and can make code more confusing, so we encourage you to limit them to where they are truly useful. For example, in the preceding examples, returning the doubled value would have worked just as well and would have been easier to understand than using a reference parameter.

Scope

Every variable has a *scope*, or a range of the program where it is accessible. Variables declared outside of any function have *global scope* and can be seen throughout the program. Variables defined inside a function have *local scope* and exist only in that function. Unlike in other languages such as Java, PHP has no narrower scope than function-level. For example, if you declare a variable inside an `if` statement or loop, the variable is destroyed not when that loop ends but when the function returns. This can sometimes lead to bugs where variables are still alive that you assume would have disappeared. Example 5.42 shows a brief example of two variables whose values live on to be printed at the end of a function.

```
function scope_example() {
  for ($i = 0; $i < 10; $i++) {
    print "Hello\n";
    $x = 42;
  }

  # $i and $x are still alive here
  print "i = $i, x = $x\n";    # i = 10, x = 42
}
```

Example 5.42 Function-level scope of variables

To avoid name collisions between local and global variables, PHP requires functions to explicitly declare when they want to access global variables. Otherwise any time a function's code refers to a variable, even one that has not yet been declared, PHP treats the variable as local. To access a global variable within a function, use a global declaration statement at the top of the function's body, using the syntax shown in Example 5.43.

```
global $variableName;
```

Example 5.43 Syntax template for global variable access

In Example 5.44, the code accesses and modifies the value of a global variable named `$show`. Without the `global` declaration, `$show` would not be visible inside the `downgrade` function.

```
$show = "Star Trek";            # global

function downgrade() {
  global $show;
  $suffix = " Voyager";         # local
  $show = "$show $suffix";
  print "$show\n";
}
```

Example 5.44 Global variable

Referring to globals should not be used as a substitute for proper parameter passing. Normally globals contain important values meant to be considered constants to be used throughout your code.

Also note that variable scope is completely unrelated to the start and end of PHP `<?php` ... `?>` blocks, so a variable declared in an earlier PHP block is also visible in later PHP blocks in the same page, as shown in Example 5.45. In that example, the variable `$firstname` is declared in the first PHP block, and then it is used successfully in a later block.

```
<?php
$firstname = "Victoria";
?>

<p>Hello, world!</p>

<?php
# $firstname is still in scope here
$fullname = "$firstname Kirst";
?>

<p>Your full name is <?= $fullname ?></p>
```

```
Hello, world!

Your full name is Victoria Kirst
```

Example 5.45 Scope across PHP blocks

5.4.2 Including Files

PHP has a function named `include` that you can use to inject a file's contents into your page. If the injected file's contents are HTML code, the HTML is displayed on your page. If its contents are PHP code, the code will be executed, and any variables or functions it declares will now be available to any of your subsequent code. The following is the syntax for the `include` function:

```
include("filename");
```

Example 5.46 Syntax template for `include` statement

You can use `include` as a powerful way to eliminate redundancy at the file level. For example, you may have several pages that have a common header or share a common large block of content. Or you may have written useful utility functions and code that you'd like to call in several pages. In either case, you can place the common content into a separate file and have each page include it.

Suppose you have two pieces of important HTML and PHP code whose content you want to include in several pages on your overall web site. The first reused piece of code is a quote from a poem, and a function called `pigLatin` that you want to call in several pages. You can put this partial web content into a file such as partial1.php shown in Example 5.47.

```
<blockquote><p>
  I think that I shall never see <br /> A poem lovely as a tree
</p></blockquote>

<?php
function pigLatin($word) {
  return substr($word, 1) . "-" . $word[0] . "ay";
}
?>
```

Example 5.47 Partial web page, `partial1.php`