

7

Using Arrays

The next—and last—variable type you’ll learn about in this book is the *array*. Arrays are significantly different from numbers or strings, and you can’t make the most of programming in PHP without understanding them.

Because of their unique nature, this chapter will cover arrays more deliberately and slowly than the other variable types. The chapter begins with an introduction to the concept, along with the basics of creating and using arrays. Then it covers multidimensional arrays and some of the array-related functions. The chapter concludes with array-string conversions and a demonstration on how to create an array from an HTML form.

In This Chapter

What Is an Array?	152
Creating an Array	154
Adding Items to an Array	158
Accessing Array Elements	161
Creating Multidimensional Arrays	164
Sorting Arrays	168
Transforming Between Strings and Arrays	172
Creating an Array from a Form	176
Review and Pursue	182

Arrays store a list of things and are a way of storing data that preserves order. An example would be the days of the week or the months of the year. An array preserves the order that you put things in.

What Is an Array?

Arrays constitute a complicated but very useful notion. Whereas numbers and strings are *scalar* variables (meaning they always have only a single value), an array is a collection of multiple values assembled into one overriding variable. An array can consist of numbers and/or strings (and/or other arrays), which allows this one variable to hold exponentially more information than a simple string or number can. For example, if you wanted to create a grocery list using strings, your code would look something like this:

```
$item1 = 'apples';  
$item2 = 'bananas';  
$item3 = 'oranges';
```

For each added item, you'd need to create a new string. This approach is cumbersome, and it makes it difficult to refer back to the entire list or any specific value later in your code. You can greatly simplify matters by placing your entire list into one array (say, `$items`), which contains everything you need (Table 7.1).

As an array, your list can be added to, sorted, searched, and so forth. With this context in mind, let's look into the syntax of arrays.

In PHP, there are three types of arrays:

Indexed arrays - Arrays with numeric index

Associative arrays - Arrays with named keys

Multidimensional arrays - Arrays containing one or more arrays

http://www.w3schools.com/Php/php_arrays.asp

TABLE 7.1 Grocery List Array

Item Number	Item
1	apples
2	bananas
3	oranges

An array can store one or more elements. Each element consists of an index and a value. The index can be either an integer or a string. A value can be any PHP data type. By default, PHP uses integer indexes where (0) is the first element, 1 is the second, and so on.

**Murach's PHP and MySQL
p. 313**

An array whose keys are numbers is called an *indexed* array. If the keys are strings, it's referred to as an *associative* array.

Superglobals and You

Throughout this book, you've already dealt with some arrays: `$_SERVER`, `$_GET`, and `$_POST`. These are all special arrays called *superglobals*, along with `$_COOKIE`, `$_SESSION`, and `$_ENV`.

As you know, the `$_POST` array receives all the data sent to the page from a form that uses the POST method. Its indexes are the names of the form elements, and its values are the values of those form elements. Therefore, `$_POST['name']` refers to the value typed in a form input created by the code

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

Similarly, `$_GET` refers to data sent from a form using the GET method or from data otherwise passed in the URL. `$_COOKIE` refers to data stored in a cookie, and `$_SESSION` refers to data stored in a session (you'll encounter these two in Chapter 9, "Cookies and Sessions").

`$_ENV` is like `$_SERVER`, containing values pertaining to the computer on which PHP is running.

index value
↓ ↓
name="name"
 ↓
\$_POST['name']

Syntactical rules for arrays

The other variable types you've dealt with—numbers and strings—have a variable name and a corresponding value (for example, `$first_name` could be equal to *Larry*). Arrays also have a name, derived using the same conventions:

- They begin with a dollar sign.
- They continue with a letter or underscore.
- They finish with any combination of letters, numbers, or the underscore.

But arrays differ in that they contain multiple *elements* (think of each row in Table 7.1 as an element). An element consists of an *index* or *key* (the two words can be used interchangeably) and a value. In Table 7.1, the Item Number is the key, and the Item is the value.

An array's index is used as a reference point to the values. An array can use either numbers or strings as its keys (or both), depending on how you set it up.

Generally, when you use an array it looks the same as any other variable, except that you include a key in square brackets (`[]`) to reference particular values. Whereas `$items` refers to the array as a whole, `$items[1]` points to a specific element in the array (in this example, *apples*).

Creating an Array

The formal method of creating an array is to use the `array()` function. Its syntax is:


```
$list = array ('apples', 'bananas',  
→ 'oranges');
```

Arrays automatically begin their indexing at 0, unless otherwise specified. In that example—which doesn't specify an index for the elements—the first item, *apples*, is automatically indexed at 0, the second item at 1, and the third at 2.

You can assign the index when using `array()`:

```
$list = array (1 => 'apples', 2 =>  
→ 'bananas', 3 => 'oranges');
```

This symbol is called the association indicator.



Because PHP is very liberal when it comes to blank space in your scripts, you can make this structure easier to read by writing it over multiple lines:

```
$list = array (  
1 => 'apples',  
2 => 'bananas',  
3 => 'oranges'  
);
```

Finally, the index value you specify doesn't have to be a number—you can use strings as well. This indexing technique is practical for making more meaningful lists. As an example, you could create an array that records the soup of the day for each day of the week, as in the following script. This example will also demonstrate how you can, and cannot, print out an array (which has already been demonstrated but is worth rehashing).

Script 7.1 The `$soups` array contains three elements and uses strings for its keys.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD
  XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="Content-Type"
     content="text/html;
     charset=utf-8"/>
6   <title>No Soup for You!</title>
7 </head>
8 <body>
9   <h1>Mmm...soups</h1>
10 <?php // Script 7.1 - soups1.php
11 /* This script creates and prints out
    an array. */
12 // Address error management, if you
    want.
13
14 // Create the array:
15 $soups = array (
16   'Monday' => 'Clam Chowder',
17   'Tuesday' => 'White Chicken Chili',
18   'Wednesday' => 'Vegetarian'
19 );
20
21 // Try to print the array:
22 print "<p>$soups</p>";
23
24 // Print the contents of the array:
25 print_r ($soups);
26
27 ?>
28 </body>
29 </html>
```

Associative arrays use a string as the index for the value that's stored in the array. When using an associative array, the index is commonly called a key.

To create an array:

1. Begin a new document in your text editor or IDE, to be named **soups1.php** (Script 7.1):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
    <title>No Soup for You!</title>
</head>
<body>
<h1>Mmm...soups</h1>
```

2. Begin the PHP section of the script and address error handling, if necessary:

```
<?php // Script 7.1 - soups1.php
```

If you don't have **display_errors** enabled, or if **error_reporting** is set to the wrong level, see Chapter 3, "HTML Forms and PHP," for the lines to include here to alter those settings.

3. Use the **array()** function to create an array:

```
$soups = array (
    'Monday' => 'Clam Chowder',
    'Tuesday' => 'White Chicken Chili',
    'Wednesday' => 'Vegetarian'
);
```

continues on next page

To initialize an array with the names as the indices instead of numbers. In this case you are telling PHP that the array `$soups` should have an index of 'Monday' and a value of 'Clam Chowder'.

This is the proper format for initializing (creating and assigning a value to) an array in PHP, using strings as the indices. Because both the keys and values are strings, you surround them with quotation marks. As with all strings, you can use either single or double quotation marks, so long as you're mindful of other quotation marks that might be found within the string.

4. Attempt to print the array:

```
print "<p>$soups</p>";
```

As you've already seen, arrays are also different in that they can't be printed the way you'd print other (scalar) variables.

5. Use the `print_r()` function to print out the array differently:

```
print_r ($soups);
```

In Chapter 2, "Variables," you learned how to use `print_r()` to show the contents and structure of any variable. Use it here so that you can see the difference between the way this function and `print` work with arrays.

6. Close the PHP and the HTML sections:

```
?>
</body>
</html>
```

7. Save your document as `soups1.php`.

place it in the proper directory for your PHP-enabled server, and test it in your Web browser **A**.

Remember to run the PHP script through a URL.

Mmm...soups

Array

Array ([Monday] => Clam Chowder [Tuesday] => White Chicken Chili [Wednesday] => Vegetarian)

A Because an array is structured differently than other variable types, a request to print an array results in the word *Array*. On the other hand, the `print_r()` function prints the array's contents and structure.

TIP The practice of beginning any index at 0 is standard in PHP and most other programming languages. As unnatural as this counting system may seem, it's here to stay, so you have two possible coping techniques. First, manually start all your arrays indexed at position 1. Second, unlearn a lifetime of counting from 1. You can decide which is easier, but most programmers just get used to this odd construct.

TIP You must refer to an array's elements via the same index used to create the array. In the `$soups` example, `$soups[0]` has no value even though the array obviously has a first element (the first element normally being indexed at 0 numerically).

TIP If you use the `array()` function to define an index, you can associate the first index, and the others will follow sequentially. For example:

```
$list = array (1 => 'apples',  
              'bananas', 'oranges');
```

Now *bananas* is indexed at 2 and *oranges* at 3.

TIP The `range()` function can also be used to create an array of items based on a range of values. Here are two examples:

```
$ten = range (1, 10);  
$alphabet = range ('a', 'z');
```

TIP As of PHP version 5, the `range()` function includes a *step* parameter that lets you specify increments:

```
$evens = range (0, 100, 2);
```

TIP If you use the `var_dump()` function in your script in lieu of `print_r()`, it shows not only the contents of the array but also its structure in a more detailed format **B**.

TIP An array whose keys are numbers is called an *indexed* array. If the keys are strings, it's referred to as an *associative* array. Other languages refer to associative arrays as *hashes*.

```
array(3) { ["Monday"]=> string(12) "Clam Chowder" ["Tuesday"]=>  
string(19) "White Chicken Chili" ["Wednesday"]=> string(10)  
"Vegetarian" }
```

B The `var_dump()` function (used with Script 71 instead of the `print_r()` function) shows how many elements are in an array and how long each string value is.

Adding Items to an Array

In PHP, once an array exists, you can add extra elements to the array with the assignment operator (the equals sign), in a way similar to how you assign a value to a string or a number. When doing so, you can specify the key of the added element or not specify it, but in either case, you must refer to the array with the square brackets. To add two items to the existing `$list` array, you'd write

```
$list[] = 'pears';  
$list[] = 'tomatoes';
```

If you don't specify the key, each element is appended to the existing array, indexed with the next sequential number. Assuming this is the same array from the preceding section, which was indexed at 1, 2, and 3, *pears* is now located at 4 and *tomatoes* at 5.

If you do specify the index, the value is assigned at that location. Any existing value already indexed at that point is overwritten, like so:

```
$list[3] = 'pears';  
$list[4] = 'tomatoes';
```

Now, the value of the element in the fourth position of the array is *tomatoes*, and no element of `$list` is equal to *oranges* (that value was overwritten by *pears*). With this in mind, unless you intend to overwrite any existing data, you'll be better off not naming a specific key when adding values to your arrays. However, if the array uses strings for indices, you'll probably want to specify keys so that you don't end up with an unusual combination of string and numeric keys.

To test this process, in the following task you'll rewrite `soups1.php` to add more elements to the array. To see the difference adding more elements makes, you'll print out the number of elements in the array before and after the new additions. Just as you can find the length of a string—how

Deleting Arrays and Array Elements

You won't frequently need to delete an individual item from an array, but it's possible to do using the `unset()` function. This function eliminates a variable and frees up the memory it used. When applied to an array element, that element is deleted:

```
unset($array[4]);  
unset($array['name']);
```

If you apply `unset()` to an entire array or any other variable type, the whole variable is deleted:

```
unset($array);  
unset($string);
```

You can also *reset* an array (empty it without deleting the variable altogether) using the `array()` function:

```
$array = array();
```

This has the effect of initializing the variable: making it exist and defining its type without assigning a value.

Script 7.2 You can directly add elements to an array one at a time by assigning each element a value with the assignment operator. The **count()** function will help you keep track of how many elements the array contains.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD
   XHTML 1.0 Transitional//EN"
2    "http://www.w3.org/TR/xhtml1/DTD/
   xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
   xhtml" xml:lang="en" lang="en">
4  <head>
5    <meta http-equiv="Content-Type"
   content="text/html; charset=utf-8"/>
6    <title>No Soup for You!</title>
7  </head>
8  <body>
9    <h1>Mmm...soups</h1>
10   <?php // Script 7.2 - soups2.php
11   /* This script creates and prints out
   an array. */
12   // Address error management, if you want.
13
14   // Create the array:
15   $soups = array (
16     'Monday' => 'Clam Chowder',
17     'Tuesday' => 'White Chicken Chili',
18     'Wednesday' => 'Vegetarian'
19   );
20
21   // Count and print the current number
   of elements:
22   $count1 = count ($soups);
23   print "<p>The soups array originally
   had $count1 elements.</p>";
24
25   // Add three items to the array:
26   $soups['Thursday'] = 'Chicken
   Noodle';
27   $soups['Friday'] = 'Tomato';
28   $soups['Saturday'] = 'Cream of Broccoli';
29
30   // Count and print the number of
   elements again:
31   $count2 = count ($soups);
32   print "<p>After adding 3 more soups,
   the array now has $count2 elements.</p>";
33
34   // Print the contents of the array:
35   print_r ($soups);
36
37   ?>
38 </body>
39 </html>
```

many characters it contains—by using **strlen()**, you can determine the number of elements in an array by using **count()**:

\$show_many = count(\$array);

To add elements to an array:

1. Open **soups1.php** in your text editor or IDE, if it is not already.
2. After the array is initialized using **array()**, add the following (Script 7.2, to be named **soups2.php**):
\$count1 = count (\$soups);
print "<p>The soups array originally
→ had \$count1 elements.</p>";
3. Add three more elements to the array:
\$soups['Thursday'] = 'Chicken Noodle';
\$soups['Friday'] = 'Tomato';
\$soups['Saturday'] = 'Cream of
→ Broccoli';

This code adds three more soups—indexed at **Thursday**, **Friday**, and **Saturday**—to the existing array.

4. Recount how many elements are in the array, and print this value:
\$count2 = count (\$soups);
print "<p>After adding 3 more
→ soups, the array now has
→ \$count2 elements.</p>";
 5. Delete this line:
print "<p>\$soups</p>";
- This line isn't needed anymore, so you can get rid of it (you now know that you can't print an array that easily).

continues on next page

6. Save your script as **soups2.php**, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **A**.

Merging Arrays

PHP has a function that allows you to append one array onto another. Think of it as concatenation for arrays. The function, **array_merge()**, works like so:

```
$new_array = array_merge  
    ( $array1, $array2 );
```

You could also write the **soups2.php** page using this function:

```
$soups2 = array (
    'Thursday' => 'Chicken Noodle',
    'Friday'   => 'Tomato',
    'Saturday' => 'Cream of
    - Broccoli'
);
$soups = array_merge($soups,
    - $soups2);
```

You could even accomplish this result with the plus sign (thus adding two arrays together):

```
$soups = $soups + $soups2;
or
$soups += $soups2;
```

TIP Be very careful when you directly add elements to an array. There's a correct way to do it—

```
$array[] = 'Add This';
```

or

```
$array[1] = 'Add This';
```

—and an incorrect way:

```
$array = 'Add This';
```

If you forget to use the brackets, the new value will replace the entire existing array, leaving you with a simple string or number.

TIP The code

```
$array[] = 'Value';
```

creates the **\$array** variable if it doesn't yet exist.

TIP While working with these arrays, I'm using single quotation marks to enclose both the keys and the values. Nothing needs to be interpolated (like a variable), so double quotation marks aren't required. It's perfectly acceptable to use double quotation marks, though, if you want to.

TIP You don't (and, in fact, shouldn't) quote your keys if they're numbers, variables, or constants (you'll learn about constants in Chapter 8, "Creating Web Applications"). For example:

```
$day = 'Sunday';
$soups[$day] = 'Mushroom';
```

TIP The **sizeof()** function is an alias to **count()**. It also returns the number of elements in an array.

Mmm...soups

The soups array originally had 3 elements.

After adding 3 more soups, the array now has 6 elements.

```
Array ( [Monday] => Clam Chowder [Tuesday] => White Chicken Chili
[Wednesday] => Vegetarian [Thursday] => Chicken Noodle [Friday] => Tomato
[Saturday] => Cream of Broccoli )
```

A A direct way to ensure that the new elements were successfully added to the array is to count the number of elements before and after you make the additions.

Accessing Array Elements

Regardless of how you establish an array, there's only one way to retrieve a specific element (or value) from it, and that is to refer to its index:

```
print "The first item is $array[0]";
```

If the array uses strings for indexes, which should be quoted, you must adjust for the quotation marks you'd use around the index, because they conflict with the **print** syntax. This line will cause problems **A**:

```
print "<p>Monday's soup is  
→ $soups['Monday'].</p>";
```

To combat this issue, you can wrap the whole array construct within curly brackets **B**:

```
print "<p>Monday's soup is  
→ {$soups['Monday']}</p>";
```

Ironically, the feature that makes arrays so useful—being able to store multiple values in one variable—also gives it a limitation

the other variable types don't have: You must know the keys of the array in order to access its elements. If the array was set using strings, like the **\$soups** array, then referring to **\$soups[1]** points to nothing **C**. For that matter, because *indexes are case-sensitive*, **\$soups['monday']** is meaningless because **Clam Chowder** was indexed at **\$soups['Monday']**.

The fastest and easiest way to access all the values of an array is to use a **foreach** loop. This construct loops through every element of an array:

```
foreach ($array as $key => $value) {  
    print "<p>Key is $key. Value  
→ is $value</p>";  
}
```

With each iteration of the loop, the current array element's key will be assigned to the **\$key** variable and the value to **\$value**. Note that you can use any variable here: **\$k** and **\$v** are likely choices, too.

You can now write a new soups script to use this knowledge.

Parse error: syntax error, unexpected T_ENCAPSED_AND_WHITESPACE, expecting T_STRING or T_VARIABLE or T_NUM_STRING in /Users/larryullman/Sites/phpvqs4/soups2.php on line 37

A Referencing a specific element in an associative array will cause parse errors within double quotation marks.

Monday's soup is Clam Chowder.

B Wrapping an array element reference in curly brackets is one way to avoid parse errors.

Notice: Undefined offset: 1 in /Users/larryullman/Sites/phpvqs4/soups2.php on line 37

C Referring to an array index that does not exist will create an *Undefined offset* or *Undefined Index* notice.

To print the values of any array:

1. Begin a new document in your text editor or IDE (Script 7.3, to be named **soups3.php**):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
    <title>No Soup for You!</title>
</head>
<body>
<h1>Mmm...soups</h1>
```

2. Start the PHP section of the page and address error management, if you need:

```
<?php // Script 7.3 - soups3.php
```

3. Create the **\$soups** array:

```
$soups = array (
    'Monday' => 'Clam Chowder',
    'Tuesday' => 'White Chicken Chili',
    'Wednesday' => 'Vegetarian',
    'Thursday' => 'Chicken Noodle',
    'Friday' => 'Tomato',
    'Saturday' => 'Cream of Broccoli'
);
```

Here the entire array is created at once, although you could use the same method (creating the array in steps) as in the preceding script, if you'd rather.

Script 7.3 A **foreach** loop is the easiest way to access every element in an array.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD
    XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/
        xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
    xhtml" xml:lang="en" lang="en">
4  <head>
5      <meta http-equiv="Content-Type"
        content="text/html;
        charset=utf-8"/>
6      <title>No Soup for You!</title>
7  </head>
8  <body>
9  <h1>Mmm...soups</h1>
10 <?php // Script 7.3 - soups3.php
11 /* This script creates and prints out
    an array. */
12
13 // Address error management, if you
    want.
14
15 // Create the array:
16 $soups = array (
17     'Monday' => 'Clam Chowder',
18     'Tuesday' => 'White Chicken Chili',
19     'Wednesday' => 'Vegetarian',
20     'Thursday' => 'Chicken Noodle',
21     'Friday' => 'Tomato',
22     'Saturday' => 'Cream of Broccoli'
23 );
24
25 // Print each key and value:
26 foreach ($soups as $day => $soup) {
27     print "<p>$day: $soup</p>\n";
28 }
29
30 ?>
31 </body>
32 </html>
```

4. Create a **foreach** loop to print out each day's soup:

```
foreach ($soups as $day => $soup)
{
    print "<p>$day: $soup</p>\n";
}
```

The **foreach** loop iterates through every element of the **\$soups** array, assigning each index to **\$day** and each value to **\$soup**. These values are then printed out within HTML paragraph tags. The **print** statement concludes with a newline character (created by **\n**), which will make the HTML source code of the page more legible.

5. Close the PHP section and the HTML page:

```
?>
</body>
</html>
```

6. Save the page as **soups3.php**, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **D**.

Mmm...soups

Monday: Clam Chowder

Tuesday: White Chicken Chili

Wednesday: Vegetarian

Thursday: Chicken Noodle

Friday: Tomato

Saturday: Cream of Broccoli

TIP One option for working with arrays is to assign a specific element's value to a separate variable using the assignment operator:

```
$total = $array[1];
```

By doing this, you can preserve the original value in the array and still manipulate the value separately as a variable.

TIP If you only need to access an array's values (and not its keys), you can use this **foreach** structure:

```
foreach ($array as $value) {
    // Do whatever.
}
```

TIP Another way to access all of an array's elements is to use a **for** loop:

```
for ($n = 0; $n < count($array);
    $n++) {
    print "The value is $array[$n]";
}
```

TIP The curly brackets are used to avoid errors when printing array values that have strings for keys. Here are two examples where using quotation marks is not problematic, so the curly brackets aren't required:

```
$name = trim ($array['name']);
$total = $_POST['qty'] *
$_POST['price'];
```

TIP Curly brackets can also be used to separate a variable reference from a dollar sign or other characters:

```
print "The total is ${$total}.";
```

A **for** loop is commonly used for processing arrays with integer indexes, and a **foreach** loop is commonly used for processing associative arrays.

D The execution of the loop for every element in the array generates this page. The **foreach** construct allows the script to access each key and value without prior knowledge of what they are.

Creating Multidimensional Arrays

Multidimensional arrays are both simple and complicated at the same time. The structure and concept may be somewhat difficult to grasp, but creating and accessing multidimensional arrays in PHP is surprisingly easy.

You use a multidimensional array to create an array containing more information than a standard array. You accomplish this by using other arrays for values instead of just strings and numbers. For example:

```
$fruits = array ('apples', 'bananas',  
→ 'oranges');  
$meats = array ('steaks',  
→ 'hamburgers', 'pork chops');  
$groceries = array (  
  'fruits' => $fruits,  
  'meats' => $meats,  
  'other' => 'peanuts',  
  'cash' => 30.00  
);
```

This array, **\$groceries**, now consists of one string (*peanuts*), one floating-point number (30.00), and two arrays (**\$fruits** and **\$meats**).

Pointing to an element in an array within an array can seem tricky. The key (pardon the pun) is to continue adding indices in square brackets as necessary, working from the outer array inward. With that example, **bananas** is at **\$groceries['fruits'][1]**. First, you point to the element (in this case, an array) in the **\$groceries** array by using **['fruits']**. Then, you point to the element in that array based on its position—it's the second item, so you use the index **[1]**.

In this next task, you'll write a script that creates another multidimensional array example.

Script 7.4 The multidimensional **\$books** array stores a lot of information in one big variable.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD  
   XHTML 1.0 Transitional//EN"  
2      "http://www.w3.org/TR/xhtml1/DTD/  
   xhtml1-transitional.dtd">  
3  <html xmlns="http://www.w3.org/1999/  
   xhtml" xml:lang="en" lang="en">  
4  <head>  
5      <meta http-equiv="Content-Type"  
        content="text/html;  
        charset=utf-8"/>  
6      <title>Larry Ullman's Books and  
        Chapters</title>  
7  </head>  
8  <body>  
9  <h1>Some of Larry Ullman's Books</h1>  
10 <?php // Script 7.4 - books.php  
11 /* This script creates and prints out  
   a multidimensional array. */  
12 // Address error management, if you  
   want.  
13  
14 // Create the first array:  
15 $phpvqs = array (1 => 'Getting Started  
   with PHP', 'Variables', 'HTML Forms  
   and PHP', 'Using Numbers');  
16  
17 // Create the second array:  
18 $phpadv = array (1 => 'Advanced  
   PHP Techniques', 'Developing Web  
   Applications', 'Advanced Database  
   Concepts', 'Security Techniques');  
19  
20 // Create the third array:  
21 $phpmysql = array (1 => 'Introduction  
   to PHP', 'Programming with PHP',  
   'Creating Dynamic Web Sites',  
   'Introduction to MySQL');  
22  
23 // Create the multidimensional array:  
24 $books = array (  
25   'PHP VQS' => $phpvqs,  
26   'PHP Advanced VQP' => $phpadv,  
27   'PHP and MySQL VQP' => $phpmysql  
28 );  
29
```

code continues on next page

Script 7.4 *continued*

```
30 // Print out some values:
31 print "<p>The third chapter of
my first book is <i>{$books['PHP
VQS'][3]}</i>.</p>";
32 print "<p>The first chapter of my
second book is <i>{$books['PHP
Advanced VQP'][1]}</i>.</p>";
33 print "<p>The fourth chapter of
my fourth book is <i>{$books['PHP
and MySQL VQP'][4]}</i>.</p>";
34
35 // See what happens with foreach:
36 foreach ($books as $key => $value) {
37     print "<p>$key: $value</p>\n";
38 }
39
40 ?>
41 </body>
42 </html>
```

To use multidimensional arrays:

1. Begin a new document in your text editor or IDE, to be named **books.php** (Script 7.4):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/
→ DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type"
→ content="text/html;
→ charset=utf-8"/>
    <title>Larry Ullman's Books and
→ Chapters</title>
</head>
<body>
<h1>Some of Larry Ullman's
→ Books</h1>
```

2. Create the initial PHP tags, and address error management, if necessary:

```
<?php // Script 7.4 - books.php
```

3. Create the first array:

```
$phpvqs = array (1 => 'Getting
→ Started with PHP', 'Variables',
→ 'HTML Forms and PHP', 'Using
→ Numbers');
```

To build up the multidimensional array, you'll create three standard arrays and then use them as the values for the larger array. This array (called **\$phpvqs**, which is short for *PHP for the Web: Visual QuickStart Guide*) uses numbers for the keys and strings for the values. The numbers begin with 1 and correspond to the chapter numbers. The values are the chapter titles.

continues on next page

4. Create the next two arrays:

```
$phpadv = array (1 => 'Advanced
→ PHP Techniques', 'Developing Web
→ Applications', 'Advanced
→ Database Concepts', 'Security
→ Techniques');
$phpmysql = array (1 =>
→ 'Introduction to PHP',
→ 'Programming with PHP',
→ 'Creating Dynamic Web Sites',
→ 'Introduction to MySQL');
```

For each array, add only the book's first four chapters for simplicity's sake.

5. Create the main, multidimensional array:

```
$books = array (
'PHP VQS' => $phpvqs,
'PHP Advanced VQP' => $phpadv,
'PHP and MySQL VQP' => $phpmysql
);
```

The **\$books** array is the master array for this script. It uses strings for keys (which are shortened versions of the book titles) and arrays for values. Use the **array()** function to create it, as you would any other array.

6. Print out the name of the third chapter of the *PHP Visual QuickStart Guide* book:

```
print "<p>The third chapter of
→ my first book is <i>{$books['PHP
→ VQS'][3]}</i>.</p>";
```

Following the rules stated earlier, all you need to do to access any individual chapter name is to begin with **\$books**, follow that with the first index (**['PHP VQS']**), and follow that with the next index (**[3]**). Because you're placing this in a **print** call, you enclose the whole construct in curly brackets to avoid parse errors.

7. Print out two more examples:

```
print "<p>The first chapter of
→ my second book is <i>{$books
→ ['PHP Advanced VQP'][1]}</i>.</p>";
print "<p>The fourth chapter of
→ my fourth book is <i>{$books
→ ['PHP and MySQL VQP'][4]}</i>.</p>";
```

8. Run the **\$books** array through a **foreach** loop to see the results:

```
foreach ($books as $key => $value)
{
    print "<p>$key: $value</p>\n";
}
```

Some of Larry Ullman's Books

The third chapter of my first book is *HTML Forms and PHP*.

The first chapter of my second book is *Advanced PHP Techniques*.

The fourth chapter of my fourth book is *Introduction to MySQL*.

PHP VQS: Array

PHP Advanced VQP: Array

PHP and MySQL VQP: Array

Need to
have error
reporting set
to (0) to
suppress
this notice.

A The first three lines are generated by **print** statements. The last three show the results of the **foreach** loop (and the notices come from attempting to print an array).

PHP VQS

Chapter 1 is Getting Started with PHP

Chapter 2 is Variables

Chapter 3 is HTML Forms and PHP

Chapter 4 is Using Numbers

PHP Advanced VQP

Chapter 1 is Advanced PHP Techniques

Chapter 2 is Developing Web Applications

Chapter 3 is Advanced Database Concepts

Chapter 4 is Security Techniques

PHP and MySQL VQP

Chapter 1 is Introduction to PHP

Chapter 2 is Programming with PHP

Chapter 3 is Creating Dynamic Web Sites

Chapter 4 is Introduction to MySQL

B One **foreach** loop within another can access every element of a two-dimensional array.

```
Array
(
    [PHP VQS] => Array
        (
            [1] => Getting Started with PHP
            [2] => Variables
            [3] => HTML Forms and PHP
            [4] => Using Numbers
        )

    [PHP Advanced VQP] => Array
        (
            [1] => Advanced PHP Techniques
            [2] => Developing Web Applications
            [3] => Advanced Database Concepts
            [4] => Security Techniques
        )

    [PHP and MySQL VQP] => Array
        (
            [1] => Introduction to PHP
            [2] => Programming with PHP
            [3] => Creating Dynamic Web Sites
            [4] => Introduction to MySQL
        )
)
```

C The **print_r()** function shows the structure and contents of the **\$books** array.

The **\$key** variable will be assigned each abbreviated book title, and the **\$value** variable ends up containing each chapter array.

9. Close the PHP section and complete the HTML page:

```
?>
</body>
</html>
```

10. Save the file as **books.php**, place it in the proper directory for your PHP-enabled server, and test it in your browser **A**.

TIP To access every element of every array, you can nest two **foreach** loops like this **B**:

```
foreach ($books as $title =>
    → $chapters) {
    print "<p>$title";
    foreach ($chapters as $number =>
        → $chapter) {
        print "<br />Chapter $number
            → is $chapter";
    }
    print '</p>';
}
```

TIP Using the **print_r()** or **var_dump()** function (preferably enclosed in HTML **<pre>** tags for better formatting), you can view an entire multidimensional array **C**.

TIP You can create a multidimensional array in one statement by using a series of nested **array()** calls (instead of using several steps as in this example). However, doing so isn't recommended, because it's all too easy to make syntactical errors as a statement becomes more and more nested.

TIP Although all the subarrays in this example have the same structure (numbers for indexes and four elements), that isn't required with multidimensional arrays.

TIP To learn about the greater "Larry Ullman Collection," including the three books referenced here, head to www.LarryUllman.com.

Sorting Arrays

PHP supports a variety of ways to sort an array (*sort* refers to an alphabetical sort if the values being sorted are strings, or a numerical sort if the values being sorted are numbers). When you're sorting an array, you must keep in mind that an array consists of pairs of *keys and values*. Thus, an array can be sorted based on the keys or the values. This is further complicated by the fact that you can sort the values and keep the corresponding keys aligned, or you can sort the values and have them be assigned new keys.

To sort the values without regard to the keys, you use **sort()**. To sort these values (again, without regard to the keys) in reverse order, you use **rsort()**. The syntax for every sorting function is:

function(\$array);

So, **sort()** and **rsort()** are used as follows:

```
sort($array);  
rsort($array);
```

To sort the values while maintaining the correlation between each value and its key, you use **asort()**. To sort the values in reverse while maintaining the key correlation, you use **arsort()**.

To sort by the keys while maintaining the correlation between the key and its value, you use **ksort()**. Conversely, **krsort()** sorts the keys in reverse. **Table 7.2** lists all these functions.

Finally, **shuffle()** randomly reorganizes the order of an array.

As an example of sorting arrays, you'll create a list of students and the grades they received on a test, and then sort this list first by grade and then by name.

TABLE 7.2 Array Sorting Functions

Function	Sorts By	Maintains Key-Values?
sort()	Values	No
rsort()	Values (inverse)	No
asort()	Values	Yes
arsort()	Values (inverse)	Yes
ksort()	Keys	Yes
krsort()	Keys (inverse)	Yes

- **sort()** - sort arrays in ascending order
- **rsort()** - sort arrays in descending order
- **asort()** - sort associative arrays in ascending order, according to the value
- **ksort()** - sort associative arrays in ascending order, according to the key

PHP Sorting Arrays

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- `sort()` - sort arrays in ascending order
- `rsort()` - sort arrays in descending order
- `asort()` - sort associative arrays in ascending order, according to the value
- `ksort()` - sort associative arrays in ascending order, according to the key
- `arsort()` - sort associative arrays in descending order, according to the value
- `krsort()` - sort associative arrays in descending order, according to the key

Sort Array in Ascending Order - `sort()`

The following example sorts the elements of the `$cars` array in ascending alphabetical order:

Example

```
<?php
$cars=array("Volvo","BMW","Toyota");
sort($cars);
?>
```

The following example sorts the elements of the `$numbers` array in ascending numerical order:

Example

```
<?php
$numbers=array(4,6,2,22,11);
sort($numbers);
?>
```

Sort Array in Descending Order - `rsort()`

The following example sorts the elements of the `$cars` array in descending alphabetical order:

Example

```
<?php
$cars=array("Volvo","BMW","Toyota");
rsort($cars);
?>
```

The following example sorts the elements of the `$numbers` array in descending numerical order:

Example

```
<?php
$numbers=array(4,6,2,22,11);
rsort($numbers);
?>
```

Sort Array in Ascending Order, According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```
<?php
$age=array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
?>
```

Sort Array in Ascending Order, According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```
<?php
$age=array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

Sort Array in Descending Order, According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```
<?php
$age=array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

Sort Array in Descending Order, According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```
<?php
$age=array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
?>
```

Script 7.5 PHP provides a number of different functions for sorting arrays, including **arsort()** and **krsort()**.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD
  XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="Content-Type"
     content="text/html;
     charset=utf-8"/>
6   <title>My Little Gradebook</title>
7 </head>
8 <body>
9 <?php // Script 7.5 - sort.php
10 /* This script creates, sorts, and
    prints out an array. */
11
12 // Address error management, if you
    want.
13
14 // Create the array:
15 $grades = array(
16 'Richard' => 95,
17 'Sherwood' => 82,
18 'Toni' => 98,
19 'Franz' => 87,
20 'Melissa' => 75,
21 'Roddy' => 85
22 );
23
24 // Print the original array:
25 print '<p>Originally the array looks
    like this: <br />';
26 foreach ($grades as $student =>
    $grade) {
27   print "$student: $grade<br />\n";
28 }
29 print '</p>';
30
31 // Sort by value in reverse order,
    then print again:
32 arsort($grades);
33 print '<p>After sorting the array by
    value using arsort(), the array looks
    like this: <br />';
```

code continues on next page

To sort an array:

1. Begin a new document in your text editor or IDE, to be named **sort.php** (Script 7.5):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/
  → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
  <meta http-equiv="Content-Type"
  → content="text/html;
  charset=utf-8"/>
  <title>My Little Gradebook</
  title>
</head>
<body>
```

2. Begin the PHP section, and address error handling, if desired:

```
<?php // Script 7.5 - sort.php
```

3. Create the array:

```
$grades = array(
  'Richard' => 95,
  'Sherwood' => 82,
  'Toni' => 98,
  'Franz' => 87,
  'Melissa' => 75,
  'Roddy' => 85
);
```

The **\$grades** array consists of six students' names along with their corresponding grades. Because the grades are numbers, they don't need to be quoted when assigning them.

continues on next page

4. Print a caption, and then print each element of the array using a **foreach** loop:

```
print '<p>Originally the array  
→ looks like this: <br />';  
foreach ($grades as $student =>  
→ $grade) {  
    print "$student: $grade<br />\n";  
}  
print '</p>';
```

As the **\$grades** array will be printed three times, captions indicating each state of the array will be useful. At first, the script prints the array in the original order. To do that, use a **foreach** loop, where each index (the student's name) is assigned to **\$student**, and each value (the student's grade) is assigned to **\$grade**. The final **print** call closes the HTML paragraph.

5. Sort the array in reverse order by value to determine who has the highest grade:

```
arsort ($grades);
```

To determine who has the highest grade, you need to use **arsort()** instead of **asort()**. The latter, which sorts the array in numeric order, would order the grades 75, 82, 85, and so on, rather than the desired 98, 95, 87.

You also must use **arsort()** and not **rsort()** in order to maintain the key-value relationship (**rsort()** would eliminate the student's name associated with each grade).

6. Print the array again (with a caption), using another loop:

```
print '<p>After sorting the array  
→ by value using arsort(), the  
→ array looks like this: <br />';  
foreach ($grades as $student =>  
→ $grade) {  
    print "$student: $grade<br />\n";  
}  
print '</p>';
```

Script 7.5 *continued*

```
34 foreach ($grades as $student =>  
    $grade) {  
35     print "$student: $grade<br />\n";  
36 }  
37 print '</p>';  
38  
39 // Sort by key, then print again:  
40 ksort ($grades);  
41 print '<p>After sorting the array by  
    key using ksort(), the array looks  
    like this: <br />';  
42 foreach ($grades as $student =>  
    $grade) {  
43     print "$student: $grade<br />\n";  
44 }  
45 print '</p>';  
46  
47 ?>  
48 </body>  
49 </html>
```

Originally the array looks like this:

Richard: 95
Sherwood: 82
Toni: 98
Franz: 87
Melissa: 75
Roddy: 85

After sorting the array by value using `arsort()`, the array looks like this:

Toni: 98
Richard: 95
Franz: 87
Roddy: 85
Sherwood: 82
Melissa: 75

After sorting the array by key using `ksort()`, the array looks like this:

Franz: 87
Melissa: 75
Richard: 95
Roddy: 85
Sherwood: 82
Toni: 98

A You can sort an array in a number of ways with varied results. Pay close attention to whether you want to maintain your key-value association when choosing a sort function.

- Sort the array by key to put the array in alphabetical order by student name:

```
ksort ($grades);
```

The **ksort()** function organizes the array by key (in this case, alphabetically) while maintaining the key-value correlation.

- Print a caption and the array one last time:

```
print '<p>After sorting the array  
→ by key using ksort(), the array  
→ looks like this: <br />';  
foreach ($grades as $student =>  
→ $grade) {  
    print "$student: $grade<br />\n";  
}  
print '</p>';
```

- Complete the script with the standard PHP and HTML tags:

```
?>  
</body>  
</html>
```

- Save your script as **sort.php**, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **A**.

TIP Because each element in an array must have its own unique key, the `$grades` array will only work using unique student names.

TIP The **natsort()** and **natcasesort()** functions sort a string (while maintaining key-value associations) using *natural order*. The most obvious example of natural order sorting is that it places *name2* before *name12*, whereas **sort()** orders them *name12* and then *name2*.

TIP The **usort()**, **uasort()**, and **ursort()** functions let you sort an array using a user-defined comparison function. These functions are most often used with multidimensional arrays.

Transforming Between Strings and Arrays

Now that you have an understanding of both strings and arrays, this next section introduces two functions for switching between the formats. The first, `implode()`, turns an array into a string. The second, `explode()`, does just the opposite. Here are some reasons to use these functions:

- To turn an array into a string in order to pass that value appended to a URL (which you can't do as easily with an array)
- To turn an array into a string in order to store that information in a database
- To turn a string into an array to convert a comma-delimited text field (say a keyword search area of a form) into its separate parts

The syntax for using `explode()` is as follows:

```
$array = explode(separator, $string);
```

The *separator* refers to whatever character(s) define where one value ends and another begins. Commonly this is a comma, a tab, or a blank space. Thus your code might be

```
$array = explode(',', $string);
```

or

```
$array = explode(' ', $string);
```

To go from an array to a string, you need to define what the separator (aka the *glue*) should be, and PHP does the rest:

```
$string = implode(glue, $array);
```

```
$string = implode(',', $array);
```

or

```
$string = implode(' ', $array);
```

To demonstrate how to use `explode()` and `implode()`, you'll create an HTML form that takes a space-delimited string of names from the user **A**. The PHP script will then turn the string into an array so that it can sort the list. Finally, the code will create and return the alphabetized string **B**.

Enter the words you want alphabetized with each individual word separated by a space:

Alphabetize!

A This HTML form takes a list of words, which is then alphabetized by the `list.php` script **B**.

An alphabetized version of your list is:

Allison
Brian
Eric
Mark
Mike
Shauna
Sommar

B Here's the same list, alphabetized for the user. This process is quick and easy to code, but doing so would be impossible without arrays.

Script 7.6 This is a simple HTML form where a user can submit a list of words. Including detailed instructions for how the form should be used is a prudent Web design policy.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD
   XHTML 1.0 Transitional//EN"
2    "http://www.w3.org/TR/xhtml1/DTD/
   xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
   xhtml" xml:lang="en" lang="en">
4  <head>
5    <meta http-equiv="Content-Type"
      content="text/html;
      charset=utf-8"/>
6    <title>I Must Sort This Out!</title>
7  </head>
8  <body>
9    <!-- Script 7.6 - list.html -->
10   <div><p>Enter the words you want
      alphabetized with each individual word
      separated by a space:</p>
11
12   <form action="list.php" method="post">
13
14     <input type="text" name="words"
       size="60" />
15     <input type="submit" name="submit"
       value="Alphabetize!" />
16
17   </form>
18 </div>
19 </body>
20 </html>
```

To create the HTML form:

1. Begin a new document in your text editor or IDE, to be named **list.html** (Script 7.6):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/
   → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
  <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
  <title>I Must Sort This Out!
    → </title>
</head>
<body>
  <!-- Script 7.6 - list.html -->
```

2. Create an HTML form with a text input:

```
<div><p>Enter the words you
→ want alphabetized with each
→ individual word separated by
→ a space:</p>
<form action="list.php"
→ method="post">
  <input type="text" name="words"
    → size="60" />
```

It's important in cases like this to instruct the user. For example, if the user enters a comma-delimited list, the PHP script won't be able to handle the string properly (after completing both scripts, try using commas in lieu of spaces and see what happens).

continues on next page

3. Create a submit button, and then close the form and the HTML page:

```
<input type="submit" name=
→ "submit" value="Alphabetize!" />
</form>
</div>
</body>
</html>
```

4. Save your script as **list.html** and place it in the proper directory for your PHP-enabled server.

Now you'll write the **list.php** page to process the data generated by **list.html**.

To convert between strings and arrays:

1. Begin a new document in your text editor or IDE, to be named **list.php** (Script 7.7):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
    <title>I Have This Sorted Out
    → </title>
</head>
<body>
<?php // Script 7.7 - list.php
```

2. Turn the incoming string, **\$_POST['words']**, into an array:

```
$words_array = explode(' ' ,
→ $_POST['words']);
```

This line of code creates a new array, **\$words_array**, out of the string

Script 7.7 Because the **explode()** and **implode()** functions are so simple and powerful, you can quickly and easily sort a submitted list of words (of practically any length) in just a couple of lines.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD
  XHTML 1.0 Transitional//EN"
2     "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5     <meta http-equiv="Content-Type"
      content="text/html;
      charset=utf-8"/>
6     <title>I Have This Sorted Out</title>
7 </head>
8 <body>
9 <?php // Script 7.7 - list.php
10 /* This script receives a string in
   $_POST['words']. It then turns it into
   an array,
11 sorts the array alphabetically, and
   reprints it. */
12
13 // Address error management, if you
   want.
14
15 // Turn the incoming string into an
   array:
16 $words_array = explode(' ' ,
   $_POST['words']);
17
18 // Sort the array:
19 sort($words_array);
20
21 // Turn the array back into a string:
22 $string_words = implode('<br />',
   $words_array);
23
24 // Print the results:
25 print "<p>An alphabetized
   version of your list is: <br />
   $string_words</p>";
26
27 ?>
28 </body>
29 </html>
```

The php manual website for all of the array functions.

<http://www.php.net/manual/en/ref.array.php>

Add this after the closing `?>`

```
<pre>
<?php
print_r($words_array);
?>
</pre>
```

`$_POST['words']`. Each space between the words in `$_POST['words']` indicates that the next word should be a new array element. Hence the first word becomes `$words_array[0]`, then there is a space in `$_POST['words']`, then the second word becomes `$words_array[1]`, and so forth, until the end of `$_POST['words']`.

- Sort the array alphabetically:

```
sort($words_array);
```

Because you don't need to maintain key-value associations in the `$words_array`, you can use `sort()` instead of `asort()`.

- Create a new string out of the sorted array:

```
$string_words = implode('<br />',
→ $words_array);
```

Arrays don't print as easily as strings, so turn `$words_array` into a string called `$string_words`. The resulting string starts with the value of `$words_array[0]`, followed by the HTML `
` tag, the value of `$words_array[1]`, and so on. Using `
` instead of a space or comma gives the list a more readable format when it's printed to the browser.

- Print the new string to the browser:

```
print "<p>An alphabetized
→ version of your list is:
→ <br />$string_words</p>";
```

- Close the PHP section and the HTML page:

```
>?>
</body>
</html>
```

- Save your page as **list.php**, place it in the same directory as **list.html**, and test both scripts in your Web browser

A and **B**.

TIP You'll also run across code written using the `join()` function, which is synonymous with `implode()`.

Creating an Array from a Form

Throughout this chapter, you've established arrays entirely from within a PHP page. You can, however, send an array of data to a PHP script via an HTML form. In fact, every time you use `$_POST`, this is the case. But you can take this one step further by creating arrays using an HTML form. Such arrays will then be a part of the greater `$_POST` array (thereby making `$_POST` a multidimensional array).

A logical use of this capability is in dealing with checkboxes, where users might need to select multiple options from a group **A**. The HTML source code for a checkbox is as follows:

```
<input type="checkbox" name="topping" value="Ham" />
```

The problem in this particular case is that each form element must have a unique name. If you created several checkboxes, each with a name of *topping*, only the value of the last checked box would be received in the PHP script. If you were to create unique names for each checkbox—*ham*, *tomato*, *black_olives*, etc.—working with the selected values would be tedious.

The workaround is to use array syntax, as demonstrated in the next example.

The `<label>` element does not render as anything special for the user. However, it provides a usability improvement for mouse users, because if the user clicks on the text within the `<label>` element, it toggles the control. http://www.w3schools.com/tags/tag_label.asp

Pizza Toppings: ☐ Extra Tomato ☐ Ham ☐ Sausage ☐ Pepperoni
☐ Black Olives ☐ Turnips ☐ Kumquats

A Checkboxes in an HTML form, presenting several possible options.

Script 7.8 This HTML form has an array for the checkbox input names.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD
   XHTML 1.0 Transitional//EN"
2    "http://www.w3.org/TR/xhtml1/DTD/
   xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
   xhtml" xml:lang="en" lang="en">
4    <head>
5      <meta http-equiv="Content-Type"
6        content="text/html;
7        charset=utf-8"/>
8      <title>Add an Event</title>
9    </head>
10   <body>
11     <!-- Script 7.8 - event.html -->
12     <div><p>Use this form to add an
13       event:</p>
14     <form action="event.php" method="post">
15       <p>Event Name: <input type="text"
16         name="name" size="30" /></p>
17       <p>Event Days:
18         <input type="checkbox" name=
19           "days[]" value="Sunday" /> Sun
20         <input type="checkbox" name=
21           "days[]" value="Monday" /> Mon
22         <input type="checkbox" name=
23           "days[]" value="Tuesday" /> Tue
24         <input type="checkbox" name=
25           "days[]" value="Wednesday" /> Wed
26         <input type="checkbox" name=
27           "days[]" value="Thursday" /> Thu
28         <input type="checkbox" name=
29           "days[]" value="Friday" /> Fri
30         <input type="checkbox" name=
31           "days[]" value="Saturday" /> Sat
32       </p>
33       <input type="submit" name="submit"
34         value="Add the Event!" />
35     </form>
36   </div>
37 </body>
38 </html>
```

To create an array with an HTML form:

1. Begin a new document in your text editor or IDE, to be named **event.html** (Script 7.8):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
→ DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
  <meta http-equiv="Content-Type"
→ content="text/html;
→ charset=utf-8"/>
  <title>Add an Event</title>
</head>
<body>
  <!-- Script 7.8 - event.html -->
  <div><p>Use this form to add an
→ event:</p>
```

2. Begin the HTML form:

```
<form action="event.php"
→ method="post">
```

This form will be submitted to **event.php**, found in the same directory as this HTML page.

3. Create a text input for an event name:

```
<p>Event Name: <input type="text"
→ name="name" size="30" /></p>
```

This example allows the user to enter an event name and the days of the week when it takes place.

continues on next page

4. Create the days checkboxes:

```
<p>Event Days:
<input type="checkbox" name=
→ "days[]" value="Sunday" /> Sun
<input type="checkbox" name=
→ "days[]" value="Monday" /> Mon
<input type="checkbox" name=
→ "days[]" value="Tuesday" /> Tue
<input type="checkbox" name=
→ "days[]" value="Wednesday" /> Wed
<input type="checkbox" name=
→ "days[]" value="Thursday" /> Thu
<input type="checkbox" name=
→ "days[]" value="Friday" /> Fri
<input type="checkbox" name=
→ "days[]" value="Saturday" /> Sat
</p>
```

All of these checkboxes use *days[]* as the **name** value, which creates a `$_POST['days']` array in the PHP script. The **value** attributes differ for each checkbox, corresponding to the day of the week.

5. Complete the HTML form:

```
<input type="submit" name=
→ "submit" value="Add the
→ Event!" />
</form>
```

6. Complete the HTML page:

```
</div>
</body>
</html>
```

7. Save your page as **event.html** and place it in the proper directory for your PHP-enabled server.

You also need to write the **event.php** page to handle this HTML form.

Script 7.9 This PHP script receives an array of values in `$_POST['days']`.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD
  XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="Content-Type"
     content="text/html;
     charset=utf-8"/>
6   <title>Add an Event</title>
7 </head>
8 <body>
9 <?php // Script 7.9 - event.php
10 /* This script handle the event form. */
11
12 // Address error management, if you
   want.
13
14 // Print the text:
15 print "<p>You want to add an event
   called <b>{$_POST['name']}</b> which
   takes place on: <br />";
16
17 // Print each weekday:
18 if (isset($_POST['days']) AND
   is_array($_POST['days'])) {
19
20   foreach ($_POST['days'] as $day) {
21     print "$day<br />\n";
22   }
23
24 } else {
25   print 'Please select at least one
     weekday for this event!';
26 }
27
28 // Complete the paragraph:
29 print '</p>';
30 ?>
31 </body>
32 </html>
```

When you use an associative array in an echo/print statement with quotation marks, you must enclose it in curly braces. See page 161

To handle the HTML form:

1. Begin a new document in your text editor or IDE, to be named **event.php** (Script 7.9):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/
  → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org
→ /1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
  <title>Add an Event</title>
</head>
<body>
```

2. Create the initial PHP tag, address error management (if need be), and print an introductory message:

```
<?php // Script 7.9 - event.php
print "<p>You want to add an event
→ called <b>{$_POST['name']}</b>
→ which takes place on: <br />";
```

The **print** line prints out the value of the event's name. In a real-world version of this script, you would add a conditional to check that a name value was entered first (see Chapter 6, "Control Structures").

3. Begin a conditional to check that at least one weekday was selected:

```
if (isset($_POST['days']) AND
→ is_array($_POST['days'])) {
```

If no checkbox was clicked, then `$_POST['days']` won't be an existing variable. To avoid an error caused by referring to a variable that does not exist, the first part of the conditional checks that `$_POST['days']` is set.

continues on next page

The second part of the condition—and both must be **TRUE** for the entire condition to be **TRUE**—confirms that `$_POST['days']` is an array. This is a good step to take because a **foreach** loop will create an error if it receives a variable that isn't an array **B**.

4. Print each selected weekday:

```
foreach ($_POST['days'] as $day) {  
    print "$day<br />";  
}
```

To print out each checked weekday, run the `$_POST['days']` array through a **foreach** loop. The array contains the values (from the HTML form inputs, for example, *Monday*, *Tuesday*, and so on) for every box that was selected.

5. Complete the `is_array()` conditional:

```
} else {  
    print 'Please select at least  
    → one weekday for this event!';  
}
```

If no weekday was selected, then the `isset()` AND `is_array()` condition is **FALSE**, and this message is printed.

The List Function

The `list()` function is used to assign array element values to individual variables. To start with an example:

```
$date = array('Thursday', 23,  
    → 'October');  
list($weekday, $day, $month) =  
    → $date;
```

Now there is a `$weekday` variable with a value of *Thursday*, a `$day` variable with a value of *23*, and a `$month` variable with a value of *October*.

There are two caveats for using `list()`. First, `list()` only works on arrays numerically indexed starting at 0. Second, when you're using the `list()` function, you must acknowledge each array element. You could not do this:

```
list($weekday, $month) = $date;
```

But you can use empty values to ignore elements:

```
list ($weekday, , $month) = $date;
```

or

```
list (, , $month) = $date;
```

The `list()` function is often used when retrieving values from a database.

Warning: Invalid argument supplied for foreach() in /Users/larryullman/Sites/phpvqs4/event.php on line 16

B Attempting to use **foreach** on a variable that is not an array is a common cause of errors.

6. Complete the main paragraph, the PHP section, and the HTML page:

```
print '</p>';  
?>  
</body>  
</html>
```

7. Save the page as **event.php**, place it in the same directory as **event.html**, and test both pages in your Web browser **C**, **D**, and **E**.

TIP The same technique demonstrated here can be used to allow a user to select multiple options in a drop-down menu. Just give the menu a name with a syntax like *something[]*, then the PHP script will receive every selection in `$_POST['something']`.

Use this form to add an event:

Event Name:

Event Days: ☐ Sun ☐ Mon ☒ Tue ☒ Wed ☒ Thu ☐ Fri ☐ Sat

- C** The HTML form with its checkboxes.

You want to add an event called **Training Seminar** which takes place on:

Tuesday

Wednesday

Thursday

- D** The results of the HTML form.

You want to add an event called **Training Seminar** which takes place on:

Please select at least one weekday for this event!

- E** If users don't check any of the day boxes, they'll see this message.

Review and Pursue

If you have any problems with the review questions or the pursue prompts, turn to the book's supporting forum (www.LarryUllman.com/forum/).

Review

- What's the difference between an *indexed* array and an *associative* array?
- When should you use quotation marks for an array's key or value? When shouldn't you?
- How do you print a specific array element? How do you print every element in an array?
- What happens if you don't use the square brackets when adding an element to an array?
- What function returns the number of elements in an array?
- When must you use curly brackets for printing array elements?
- What is the difference between the **sort()** and **asort()** functions? Between **sort()** and **rsort()**?
- What is the syntax for **explode()**? For **implode()**? If you don't remember, check out the PHP manual page for either function.

Pursue

- Check out the PHP manual's pages for the array-related functions. Look into some of the other available array functions. In particular I'd recommend familiarizing yourself with **array_key_exists()**, **array_search()**, and **in_array()**.
- Rewrite **soups2.php** so that it displays the number of elements in the array without using a separate variable. Hint: You'll need to concatenate the **count()** function call into the **print** statement.
- Create another script that creates and displays a multidimensional array (or some of it, anyway).
- Rewrite **list.php** so that it uses **foreach** instead of **implode()**, but still prints each sorted word on its own line in the browser. Also add some form validation so that it only attempts to parse and sort the string if it has a value.